



SWITCHING CIRCUITS
FOR ENGINEERS
SECOND
EDITION

PRENTICE-HALL
INTERNATIONAL SERIES IN ELECTRICAL ENGINEERING



SERIES

SWITCHING CIRCUITS FOR ENGINEERS

SECOND
EDITION

Mitchell P. Marcus

*A clear and concise treatment of the design
and simplification of combinational and
sequential switching circuits, with
the emphasis on the practical
rather than the abstract*

Marcus

621.316
.5.049
M

PRENTICE
HALL

W. L. EVERITT, Editor



In the second edition of this well-known work covering the design and simplification of combinational and sequential switching circuits, the author once again presents the material clearly and concisely, with heavy emphasis on the *practical* rather than the abstract.

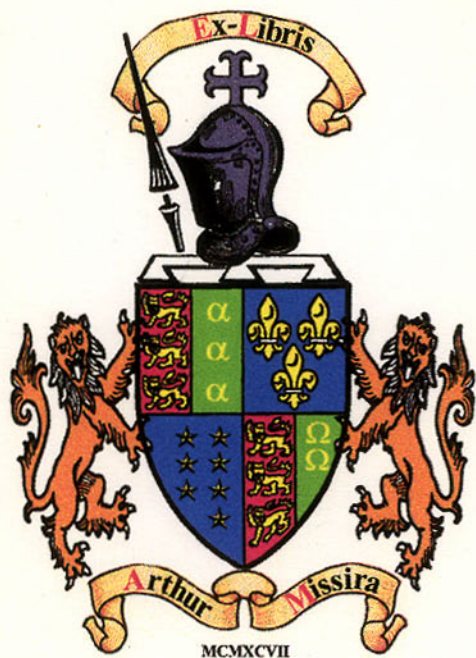
New features include: (1) Twice as many end-of-chapter problems for the reader to test his knowledge and understanding of the subject; (2) Answers or solutions to 70 per cent of these problems at the back of the book; (3) Extensive bibliography of related literature for further study; and (4) Extension of tabular method to multi-output networks.

Other outstanding features . . .

- Boolean algebra is not merely "presented" to the reader. The reader is shown exactly how to recognize the application of the theorems and is given "methods of attack."
- An original method is given for obtaining flip-flop excitation expressions in the synthesis of pulse input sequential circuits. The ad-

(Continued on back flap)





0 893 244 010



Tel. 50181

LEICESTER COLLEGES OF ART AND TECHNOLOGY

TECHNOLOGY LIBRARY

Please return this book on or before the last date stamped below. Fines will be charged on books returned after this date.

214

10. MAY 1968			
23. SEP. 1968			
26. SEP. 1969			
21. NOV. 1969			
18. DEC. 1970			
26. FEB. 1971			
-6. APR. 1973			
26. OCT. 1973			
27 FEB. 1974			
27 FEB. 1978			
10. FEB 81			

WITHDRAWN

621.316.5.049

MARCUS, M.P. Switching circuits for
engineers. 1967

2nd ed.

4/90
t2-50
Science

**SWITCHING CIRCUITS
FOR ENGINEERS**

PRENTICE-HALL ELECTRICAL ENGINEERING SERIES*

William L. Everitt, *Editor*

- ANNER *Elementary Nonlinear Electronic Circuits*
 ANNER *Elements of Television Systems*
 ARMINGTON & VOLZ *An Introduction to Electric Circuit Analysis*
 BALABANIAN *Network Synthesis*
 BARTON *Radar Systems Analysis*
 BENEDICT *Introduction to Industrial Electronics*
 BLACKWELL & KOTZEBUE *Semiconductor-Diode Parametric Amplifiers**
 BOISVERT, ROBERT, & ROBICHAUD *Signal Flow Graphs and Applications*
 BURGER & DONOVAN *Fundamentals of Silicon Integrated Device Technology*
 CARLIN & GIORDANO *Network Theory: An Introduction to Reciprocal and Nonreciprocal Circuits*
 CHANG, K. N. *Parametric and Tunnel Diodes*
 CHANG, S. *Energy Conversion**
 CHIRLIAN *Integrated and Active Network Analysis and Synthesis*
 DAVIS & WEED *Industrial Electronic Engineering*
 DEKKER *Electrical Engineering Materials*
 DEL TORO *Principles of Electrical Engineering*
 DE PIAN *Linear Active Network Theory*
 DOWNING *Modulation Systems and Noise**
 DUNN & BARKER *Electrical Measurements Manual*
 EVANS *Experiments in Electronics*
 EVANS *Introduction to Electronics*
 FETT *Feedback Control Systems*
 FICH *Transient Analysis in Electrical Engineering*
 FICH & POTTER *Theory of A-C Circuits*
 FLORES *Computer Logic: The Functional Design of Digital Computers*
 FLORES *The Logic of Computer Arithmetic**
 FOCKE *Introduction to Electrical Engineering Science*
 GENTRY, ET AL. *Semiconductor Controlled Rectifiers: Principles and Applications of p-n-p-n Devices*
 GOLDMAN *Information Theory*
 GOLDMAN *Transformation Calculus and Electrical Transients*
 GOLOMB, ET AL. *Digital Communications**
 GRAY *Digital Computer Engineering**
 HERRERO & WILLONER *Synthesis of Filters*
 HERSHBERGER *Principles of Communication Systems*
 JORDAN *Electromagnetic Waves and Radiating Systems*
 KUO *Analysis and Synthesis of Sampled-Data Control Systems**
 KUO *Automatic Control Systems, 2nd ed.*
 LE CROISSETTE *Transistors*
 LEGROS & MARTIN *Transform Calculus for Electrical Engineers*
 LO, ET AL. *Transistor Electronics*
 MALEY & EARLE *The Logic Design of Transistor Digital Computers**

MALEY & SKIKO *Modern Digital Computers*
 MARCUS *Switching Circuits for Engineers, 2nd ed.**
 MARTIN *Electronic Circuits*
 MARTIN *Physical Basis for Electrical Engineering*
 MARTIN *Ultrahigh Frequency Engineering*
 MATSCH *Capacitors, Magnetic Circuits, and Transformers*
 MOSKOWITZ & RACKER *Pulse Techniques*
 NIXON *Principles of Automatic Controls*
 NUSSBAUM *Electromagnetic and Quantum Properties of Materials*
 NUSSBAUM *Electromagnetic Theory for Engineers and Scientists*
 NUSSBAUM *Semiconductor Device Physics**
 OGATA *State Space Analysis of Control Systems*
 PARTRIDGE *Principles of Electronic Instruments*
 PASKUSZ & BUSSELL *Linear Circuit Analysis*
 PIERUSCHKA *Principles of Reliability**
 POTTER & FICH *Theory of Networks and Lines*
 PUMPHREY *Electrical Engineering, 2nd ed.*
 PUMPHREY *Fundamentals of Electrical Engineering, 2nd ed.*
 REED *Electric Networks Synthesis*
 REED *Foundation for Electric Network Theory*
 RIDEOUT *Active Networks*
 ROBERTS & VANDERSLICE *Ultrahigh Vacuum and Its Applications**
 ROBICHAUD, ET AL. *Signal Flow Graphs and Applications**
 RUSSELL *Modulation and Coding in Information Systems*
 RYDER, F. L. *Creative Engineering Analysis*
 RYDER, J. D. *Electronic Engineering Principles, 3rd ed.*
 RYDER, J. D. *Electronic Fundamentals and Applications, 3rd ed.*
 RYDER, J. D. *Networks, Lines and Fields, 2nd ed.*
 SANFORD *Physical Networks*
 SHEDD *Fundamentals of Electromagnetic Waves*
 SKRODER & HELM *Circuit Analysis by Laboratory Methods, 2nd ed.*
 SOOHOO *Theory and Application of Ferrites*
 STOUT *Basic Electrical Measurements, 2nd ed.*
 THOMSON *Laplace Transformation, 2nd ed.*
 VAN DER ZIEL *Noise*
 VAN DER ZIEL *Solid State Physical Electronics*
 VAN VALKENBURG *Network Analysis, 2nd ed.*
 VON TERSCH & SWAGO *Recurrent Electrical Transients*
 WALLMARK & JOHNSON, EDS. *Field-Effect Transistors: Physics, Technology and Applications*
 WARD *Introduction to Electrical Engineering, 2nd ed.*
 WARFIELD *Introduction to Electronic Analog Computers*
 WEED & DAVIS *Fundamentals of Electron Devices and Circuits*

* This title is also in the Prentice-Hall International Series in Electrical Engineering. Prentice-Hall, Inc.; Prentice-Hall International, Inc., United Kingdom and Eire; Prentice-Hall of Canada, Ltd., Canada.

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA, PTY. LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA (PRIVATE) LTD., *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*

MITCHELL P. MARCUS

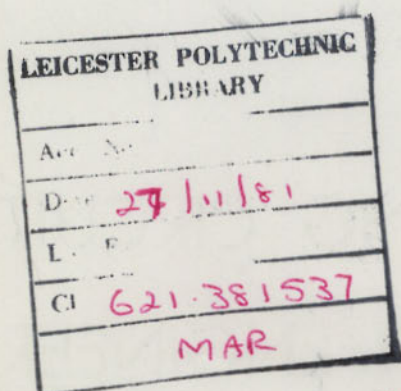
IBM Corporation
Systems Development Division
Endicott Laboratory
Endicott, New York

SWITCHING CIRCUITS FOR ENGINEERS

Second Edition

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey

To *El*
Bunny
Glenn
Lee
and *Ricky*



13 MAR 1968

621.316.5.049
M

© 1962, 1967 by PRENTICE-HALL, INC.
Englewood Cliffs, N.J.

All rights reserved. No part of this book
may be reproduced in any form or by any
means without permission in writing from
the publisher.



Current printing (last digit):
10 9 8 7 6 5 4 3 2 1

Library of Congress Catalog Card No. 67-10748

Printed in the United States of America



Preface

Switching circuits are circuits that perform logical functions. The design of switching circuits is referred to as logical design. Switching circuits can range in complexity from a simple staircase lighting circuit, controlled from both upstairs and downstairs, to a complex circuit that performs arithmetic in an electronic digital computer.

The interesting task of the logical designer is to properly interconnect basic logical circuit elements or "logic blocks" so that the resultant circuit performs a desired logical function. There are usually many different ways in which these logic blocks may be interconnected to realize a desired function; however, some of these ways require more logic blocks than others. So the logical designer's task goes further than merely realizing the desired logical function; he tries, in general, to realize the function with the most economical circuit that he can.

Although switching circuits can usually be designed intuitively, a circuit requirement does not have to become very complicated before intuitive methods can fail to yield the most economical network. With the tremendous growth of automatic digital computers and the increasing complexity of business machines and automatic control systems, logical design based on intuition has become increasingly difficult and time consuming. Fortunately, the logical designer of today has at his disposal many formalized methods for designing and simplifying switching circuits. These methods can not only lead to simpler circuits, but can considerably reduce the time required to reach a solution. Furthermore, the elimination of even a few redundant circuit components can increase the reliability of circuit operation, reducing down-time and servicing; it can make the circuit easier to understand,

ary states and the assignment of multiple secondary states to a row. The approach to secondary state assignment in this chapter is original. Chapter 17 completes the presentation of sequential circuit synthesis with a discussion of the Z-map, transient outputs, cyclic specifications, and elimination of hazards.

The synthesis of pulse-input sequential circuits is examined in Chapters 18 and 19 in which an original method for obtaining flip flop excitation expressions is presented. The advantage of this method is that excitation expressions for any type of flip flop can be read from a single map set.

The author has tried as much as possible throughout to emphasize the *practical* rather than the *abstract*. For the reader who would like to delve deeper into the subject, a bibliography of related literature for further study is included at the end of the book. The bibliography is arranged by chapter.

Problems are presented at the end of most of the chapters to give the reader an opportunity to test his knowledge and understanding of the subject. Answers to the majority of the problems are given at the back of the book; solutions or partial solutions are also included where it is felt that they would be helpful. Some of the end-of-chapter problems are identified by an asterisk; for these problems no answers are furnished at the back of the book; these problems can be used by instructors for assignments or for testing.

In addition to his work in switching circuit theory and its application to the logical design of IBM products, the author has taught courses in switching circuits at IBM since 1954, and this book is a development of these courses. The author is indebted to many IBMers: to those in IBM Education who encouraged him to write this book in the first place; to the IBM Corporation who, by making time and facilities available, made it possible for him to write it; to the IBM engineers who reviewed it and made many valuable suggestions and comments; to the IBM secretaries who, in addition to their regular work, so kindly typed it; and to the many IBM students who gave the first edition such a severe workout, helping to de-bug it by weeding out errors and who, by their probing questions and comments, contributed to many improvements in the second edition.

M. P. MARCUS

Binghamton, N. Y.

Contents

1	BOOLEAN ALGEBRA	1
	Postulates, 5	
	Some definitions, 6	
	Theorems, 7	
	Resumé of simplification theorems and "method of attack," 18	
	Additional theorems, 20	
	Summary of Boolean algebra postulates and theorems, 23	
2	SPECIAL FORMS OF BOOLEAN EXPRESSIONS	27
	Expanded sum of products, 28	
	Expanded product of sums, 28	
	"Minimum" Boolean expressions, 30	
	Minimum factored form, 33	
	Functions of n variables, 33	
3	LOGICAL CIRCUITS	36
	Logic blocks, 38	
	Negative logic, 41	
	Application of positive and negative logic, 44	
	Mixed logic, 45	
4	ELECTRONIC LOGIC BLOCKS	48
	Diode logic blocks, 48	
	Vacuum tube logic blocks, 49	
	Transistor logic blocks, 51	

5	CONTACT NETWORKS	57
	Implementation of AND, OR, and NOT functions, 59	
	Transfer contacts, 61	
	Bridge circuits, 63	
	Nonplanar networks, 63	
	Complementation of contact networks, 64	
	Multi-output contact networks, 65	
6	TABULAR METHOD OF SIMPLIFICATION	71
	Optional combinations, 71	
	Tabular method of simplification, 73	
	Optional combinations with tabular method, 79	
	Algebraic solution of final table, 80	
	Weighting of prime implications, 82	
	Simplification of final table, 83	
	Complementary approach with tabular method, 85	
	Iterative method for obtaining prime implicants, 87	
	Multi-output networks, 88	
7	MAP METHOD OF SIMPLIFICATION	97
	Complementary approach with map method, 102	
	"Method of attack," 103	
	Optional combinations with map method, 106	
	Maps of more than four variables, 106	
	Summary, 109	
	Factoring on the map, 109	
8	TREES—RELAY AND ELECTRONIC	112
	Relay trees, 112	
	Minimization of partial trees, 114	
	Electronic trees, 119	
9	SYMMETRIC FUNCTIONS	126
	Variables of symmetry, 126	
	<i>m-out-of-n</i> functions, 127	
	Boolean operations with symmetric notations, 128	
	Symmetric relay contact networks, 129	
	Detection and identification of symmetric switching functions, 137	
10	REITERATIVE NETWORKS	145
	Design of reiterative networks, 146	
	Sequence representation, 150	
	Elimination of redundant input lines, 151	

11	NUMBER SYSTEMS; ADDERS	156
	Number systems, 156	
	Binary adders, 161	
	Binary-coded-decimal adder, 165	
12	CODES, ERROR DETECTION, ERROR CORRECTION	167
	Nonchecking numeric codes, 167	
	Error detection and correction, 171	
	Single-error detection—minimum distance two codes, 174	
	Single-error correction—minimum distance three codes, 176	
	Single-error correction with double-error detection—minimum distance four codes, 179	
	Alphanumeric codes, 179	
	Cross-parity, 180	
13	SEQUENTIAL CIRCUITS I	182
	Concept of stability, 184	
	Basic sequential circuit operation, 185	
	Intuitive approach to sequential circuit synthesis, 185	
	Flow table, 190	
14	SEQUENTIAL CIRCUITS II	193
	Synthesis of sequential circuits, 193	
	Primitive flow table, 194	
	"Power-on" output state, 196	
	Elimination of redundant stable states, 197	
	Pseudo-equivalence, 200	
15	SEQUENTIAL CIRCUITS III	206
	Merged flow table; merger diagram, 206	
	Cycles, 209	
	Races, 210	
	Secondary state assignment and Y-map, 212	
	Transition map, 212	
	Y-map, 214	
16	SEQUENTIAL CIRCUITS IV	217
	Utilization of spare secondary states, 217	
	Assignment of multiple secondary states to a row, 230	
	Utilization of spare secondary states—summary, 233	

17	SEQUENTIAL CIRCUITS V	236
	Z-map, 236	
	Transient outputs; cyclic specifications, 241	
	Hazards, 243	
	Most-economical circuit considerations, 247	
18	PULSE-INPUT SEQUENTIAL CIRCUITS I	252
	Flip flops, 253	
	Pulse-input sequential circuits, 256	
	Synthesis of pulse-input sequential circuits, 258	
	Flow diagram, 259	
	Flow table, 260	
	Word statement to flow diagram and flow table, 261	
	Elimination of redundant states, 266	
	Secondary assignment, 269	
19	PULSE-INPUT SEQUENTIAL CIRCUITS II	273
	Flip flop excitation maps, 273	
	Map entries, 274	
	Reading the flip flop excitation maps, 278	
	Sequential circuit outputs, 295	
	Most-economical circuit considerations, 298	
	RELATED LITERATURE FOR FURTHER STUDY	301
	ANSWERS AND SOLUTIONS TO PROBLEMS	313
	INDEX	333

1

Boolean Algebra

Policy No. 22 may be issued only if the applicant

- 1. Has been issued Policy No. 19 and is a married male,
- or 2. Has been issued Policy No. 19 and is married and under 25,
- or 3. Has not been issued Policy No. 19 and is a married female,
- or 4. Is a male under 25,
- or 5. Is married and 25 or over.

From an XYZ Insurance Company Manual

Can you simplify the statement above? There is a great deal of redundancy in this policy statement. Using intuition only, most people will not be able to recognize all of the redundancy in as simple a statement as this one. An equivalent but simpler statement appears near the end of this chapter.

There is an algebra of logic, called Boolean algebra, which enables us to dispense with intuition and deductively simplify logical statements that are even much more complex. Boolean algebra is named after George

Boole who, in the middle 1800's, developed it. Almost a hundred years later Claude E. Shannon realized its application to the simplification of logical circuits or switching circuits.

Experience has shown that if one learns Boolean algebra with relation to its circuit implications he does not become proficient in it because he "thinks" in circuits rather than in the algebra. Experience has also shown that the study of Boolean algebra from a purely abstract point of view is not attractive to most engineers because there is no practical association for them to "hang their hat on." Study of Boolean algebra as it relates to logical statements has been found to be the most effective initial approach and this approach is followed here. Later, it will be shown how the algebra, one of the most basic tools available to the logical designer, can be used to simplify logical circuits.

$$X = 1 \text{ or else } X = 0$$

Consider basic logical statements that must be either true or false. For example: *The applicant is a male.* Letter symbols are used to represent such statements as follows.

$$X = \text{the applicant is a male}$$

It can now be said that X must be true or else X must be false. Carrying our symbolism a step further, a 1 is used to represent the "value" of a true statement, and a 0 is used to represent the "value" of a false statement. If the statement "The applicant is a male" is true, we say that the "value" of X is 1, written $X = 1$. If the statement is false, we say that the "value" of X is 0, written $X = 0$.

Thus

$$X = 1 \quad \text{or else} \quad X = 0$$

There is no numerical significance to the 1 and 0; there is only a logical significance.

Although 1 and 0 represent the truth or falsity of a statement, the prerogative is taken of saying, " X equals 1" or " X equals 0."

AND

In the reduction of compound logical statements to Boolean algebra, there are three key words of special importance: AND, OR, and NOT. First consider a compound statement made up of two basic logical statements connected by the word AND.

The applicant is a male AND the applicant is married

Making use of symbology

X = the applicant is a male
 Y = the applicant is married

The entire compound statement can now be written

$$X \text{ AND } Y$$

When is X AND Y true and when is X AND Y false? X may be true or false and Y may be true or false. Taken together, there are four possibilities: X and Y may both be true, X may be true and Y false, X may be false and Y true, or both X and Y may be false.

X AND Y is true only if X is true and Y is true. This can be tabulated as follows:

X		Y	$X \text{ AND } Y$
True	AND	True	True
True	AND	False	False
False	AND	True	False
False	AND	False	False

A “.” is used to symbolize AND. Thus, X AND Y is written $X \cdot Y$. Replacing *True* with 1, *False* with 0, and AND with “.” gives the following relationships:

X	Y	$X \cdot Y$
1	1	1
1	0	0
0	1	0
0	0	0

Although the “.” signifies multiplication in ordinary algebra, here it has only the logical AND significance. Other symbols have also been used to represent AND. Some of these are $+$, \wedge , and \cap .

OR

Now consider a compound statement made up of two basic logical statements connected by the word OR.

The applicant is a male OR the applicant is married

Substituting X and Y for the two statements, as before, gives

$$X \text{ OR } Y$$

This OR is the *inclusive* OR, that is, X OR Y means X or Y or both. (With the *exclusive* OR, X OR Y would mean either X or Y but not both.) Unless stated otherwise, OR will always be understood to mean the *inclusive* OR.

When is X OR Y true and when is X OR Y false? The same four possible combination of X and Y exist. X OR Y is true when X is true or when Y is true or when both are true. This is tabulated as follows:

X		Y	X OR Y
True	OR	True	True
True	OR	False	True
False	OR	True	True
False	OR	False	False

A “+” is used to symbolize OR. Thus, X OR Y is written $X + Y$. Replacing *True* with 1, *False* with 0, and OR with “+” gives the following relationships:

X	Y	$X + Y$
1	1	1
1	0	1
0	1	1
0	0	0

Although the “+” signifies addition in ordinary algebra, here it has only the logical OR significance. Some other symbols that have been used to represent OR are \cdot , \vee , and \cup .

NOT

Now consider the statement

The applicant is NOT a male

When is this statement true and when is it false? If the statement “The applicant is a male” is true, then the statement “The applicant is NOT a male” is false. If the statement “The applicant is a male” is false, then the statement “The applicant is NOT a male” is true. This can be tabulated very simply by letting X represent the statement “The applicant is a male,” and letting NOT X represent the statement “The applicant is NOT a male.”

X	NOT X
True	False
False	True

Many different symbols have been used to symbolize NOT. For instance, NOT X can be written as \bar{X} , X' , $1 - X$, or $\sim X$. We shall use the symbol \bar{X} to represent NOT X . There is less chance of “losing” the “bar” than of losing the “prime,” and the bar can be applied to an expression without the need of adding parentheses; that is, $\overline{X + Y}$ will be used instead of $(X + Y)'$ or $1 - (X + Y)$ or $\sim(X + Y)$.

If a statement is true only when a second statement is false, and vice versa, as in the case above, the two statements are said to be *complements* of each other. Thus, \bar{X} is the complement of X , and X is the complement of \bar{X} .

Using our symbology,

$$\begin{aligned} \text{if } X = 1, \text{ then } \bar{X} = \bar{1} = 0 \\ \text{if } X = 0, \text{ then } \bar{X} = \bar{0} = 1 \end{aligned}$$

This is summarized in the following table:

X	\bar{X}
1	0
0	1

We now have, by definition,

$$\bar{\bar{1}} = 0 \quad \text{and} \quad \bar{\bar{0}} = 1$$

It also follows that

$$\begin{aligned} \bar{\bar{1}} &= 1 \\ \bar{\bar{0}} &= 0 \\ \bar{\bar{\bar{X}}} &= X \end{aligned}$$

Postulates

Following is a summary of the results so far.

$X = 1 \text{ or else } X = 0$	
$1 \cdot 1 = 1$	$0 + 0 = 0$
$1 \cdot 0 = 0 \cdot 1 = 0$	$0 + 1 = 1 + 0 = 1$
$0 \cdot 0 = 0$	$1 + 1 = 1$
$\bar{\bar{1}} = 0$	$\bar{\bar{0}} = 1$

This summary represents the *postulates* of Boolean algebra. Based on

these postulates are many useful theorems that enable us to manipulate and simplify logical expressions.

Boolean algebra, like any other algebra, is composed of a set of symbols and a set of rules for manipulating these symbols. However, some differences between ordinary algebra and Boolean algebra should be stressed here. In ordinary algebra the letter symbols may take on a large or even an infinite number of values; in Boolean algebra they may assume only one of two possible values, 0 and 1. Thus, Boolean algebra is much simpler than ordinary algebra. In ordinary algebra the values have a *numerical* significance; in Boolean algebra, they have only a *logical* significance. Furthermore, the meanings of “.” and “+” in Boolean algebra—AND and OR—are entirely unrelated to their meanings in ordinary algebra—“times” and “plus.”

In one sense the choice of “.”, “+”, 1, and 0 is unfortunate because of the tendency to associate them with their counterparts in ordinary algebra. In another sense, the choice is advantageous because of the coincidental relationship that five of the six postulates involving the “.” and “+” bear to their meanings in ordinary algebra.

Some Definitions

The different letters in a Boolean expression are called *variables*. For example, in the expression

$$A \cdot \bar{B} + \bar{A} \cdot C + A \cdot (D + E)$$

there are five variables, A , B , C , D , and E . Each occurrence of a variable or its complement is called a *literal*. In the expression above there are seven literals. The “.” is usually omitted in writing expressions in Boolean algebra, and is implied merely by writing the literals, or factors, in juxtaposition. Thus,

$$A \cdot \bar{B} + \bar{A} \cdot C + A \cdot (D + E)$$

would normally be written

$$A\bar{B} + \bar{A}C + A(D + E)$$

The “.” is used only where additional clarity is required.

Two expressions are *equivalent* if one expression equals 1 only when the other equals 1, and one equals 0 only when the other equals 0. Two expressions are *complements* of each other if one expression equals 1 only when the other equals 0, and vice versa.

The complement of a Boolean expression is obtained by

changing all \cdot 's to $+$'s

changing all $+$'s to \cdot 's

changing all 1's to 0's

changing all 0's to 1's

and complementing each literal.

Thus, the complement of

$$1 \cdot A + \bar{B}C + 0$$

is

$$(0 + \bar{A})(B + \bar{C}) \cdot 1$$

When the first expression equals 1, the second equals 0, and vice-versa.

The *dual* of a Boolean expression is obtained by

changing all \cdot 's to $+$'s

changing all $+$'s to \cdot 's

changing all 1's to 0's

changing all 0's to 1's

but not complementing any literal.

Thus, the dual of

$$1 \cdot A + \bar{B}C + 0$$

is

$$(0 + A)(\bar{B} + C) \cdot 1$$

There is no general relationship between the "values" of dual expressions; that is, both may equal 1, both equal 0, or one may equal 1 while the other equals 0. Duals are of principal interest in the study of the Boolean postulates and theorems, and are also useful in simplification procedures, as we shall see later.

In the preceding table of postulates, the six postulates involving the " \cdot " and " $+$ " have been purposely arranged in three rows of two postulates each. Each pair of postulates may be considered as either complements or duals of each other since no literals are involved. The theorems that follow are presented in dual pairs.

Theorems

Many useful theorems, derived from the postulates, will now be studied. These theorems enable us to simplify logical expressions or transform them into other useful equivalent expressions.

$$1a. 0 \cdot X = 0$$

$$1b. 1 + X = 1$$

In ordinary algebra it is not generally possible to prove a theorem by substituting all possible values of the variables since there may be a large or an infinite number of values. In Boolean algebra, since the variables can have only two values, 0 and 1, theorems can easily be proved merely by testing their validity for all possible combinations of values of the variables involved. This type of proof is called proof by perfect induction.

Theorem 1a may be proved as follows: X must equal either 0 or 1. If $X = 0$, then $0 \cdot 0 = 0$. If $X = 1$, then $0 \cdot 1 = 0$. Thus, no matter what the value of X ,

$$0 \cdot X = 0$$

Theorem 1b can be proved in an analogous manner. However, the proof can be approached differently by first writing the theorem so that it is in complementary form to Theorem 1a. The theorem in this form would read $1 + \bar{X} = 1$. Based on the fact that every postulate has a complementary postulate, if a theorem is valid, then its complementary theorem is valid. This is so because if a theorem is true, based on certain postulates, then its complementary theorem must be true based on the complementary postulates. Thus, Theorem 1a having been proved, the complementary theorem $1 + \bar{X} = 1$ must also be true.

Since the validity of a theorem is based upon its being true for all possible combinations of values of the variables, there is no reason why the \bar{X} in the theorem cannot be replaced by an X . Thus, if the theorem $1 + \bar{X} = 1$ holds for all values of \bar{X} , the theorem $1 + X = 1$ must be true for all values of X . Therefore, inconsequential "bars" over the variables are omitted in the theorems. The expression $1 + X = 1$ is the dual of the expression $0 \cdot X = 0$. Thus, if a theorem is valid, then its dual theorem must also be valid. For this reason, it is not necessary to go through the mechanics of proving both of a pair of dual theorems to be true; proving one is sufficient.

The literals in a theorem may represent not only single variables but also terms or factors or longer expressions. For example, using Theorem 1,

$$0 \cdot (A\bar{B} + C) = 0 \quad 1 + A\bar{B} + C = 1$$

The important point to remember about Theorem 1 is that

$$0 \cdot \text{anything} = 0$$

and

$$1 + \text{anything} = 1$$

$$2a. 1 \cdot X = X$$

$$2b. 0 + X = X$$

This pair of theorems can be proved as easily as the first pair by sub-

stituting both possible values for X . The important point to remember about Theorem 2 is that

multiplication by 1

or

addition of 0

does not affect an expression.¹ For example,

$$1 \cdot (A\bar{B} + C) = A\bar{B} + C \quad 0 + A\bar{B} + C = A\bar{B} + C$$

3a. $XX = X$

3b. $X + X = X$

An example of the application of Theorem 3 follows.

$$(A\bar{B}\bar{B} + C)(A\bar{B} + C + C) = (A\bar{B} + C)(A\bar{B} + C) = A\bar{B} + C$$

This example stresses again that the literals in these theorems may represent not only single variables but also more complex expressions.

4a. $X\bar{X} = 0$

4b. $X + \bar{X} = 1$

If $X = 1$, then $\bar{X} = 0$; if $X = 0$, then $\bar{X} = 1$. In either case, Theorem 4a represents the product of 1 and 0, which is 0, whereas Theorem 4b represents the sum of 1 and 0, which is 1. Theorem 4 says that

anything multiplied by its complement = 0

and

anything added to its complement = 1

Some simple exercises on Theorems 1 through 4 follow. These exercises should be tried before reading the solutions that follow.

Simplify

(a) AAB

(b) $A\bar{A}B$

(c) $A\bar{A} + B$

(d) $A + A + B$

(e) $A + \bar{A} + B$

(f) $(A + \bar{A})B$

Solutions:

(a) $AAB = AB$

(b) $A\bar{A}B = 0 \cdot B = 0$

(c) $A\bar{A} + B = 0 + B = B$

(d) $A + A + B = A + B$

(e) $A + \bar{A} + B = 1 + B = 1$

(f) $(A + \bar{A})B = 1 \cdot B = B$

¹The prerogative is taken of using the terms "multiplication" and "addition" to represent the AND (\cdot) operation and OR ($+$) operation respectively. The term "product" is used to represent the result of the AND operation. Thus, XYZ is called the product of X , Y , and Z ; $(A + B)(C + D)$ is called the product of $A + B$ and $C + D$. The term "sum" is used to represent the result of the OR operation. Thus, $X + Y + Z$ is called the sum of X , Y , and Z ; $AB + CD$ is called the sum of AB and CD . Furthermore, an expression such as $(A + B)(C + D)$ is called a "product of sums," and an expression such as $AB + CD$ is called a "sum of products."

$$5a. XY = YX$$

$$5b. X + Y = Y + X$$

EXAMPLE:

$$A\bar{B} + C = \bar{B}A + C = C + A\bar{B} = C + \bar{B}A$$

$$6a. XYZ = X(YZ) = (XY)Z$$

$$6b. X + Y + Z = X + (Y + Z) \\ = (X + Y) + Z$$

$$7a. \overline{XY \dots Z} = \bar{X} + \bar{Y} + \dots + \bar{Z} \quad 7b. \overline{X + Y + \dots + Z} = \bar{X}\bar{Y} \dots \bar{Z}$$

This theorem is known as DeMorgan's theorem. Theorem 7a can be proved as follows: If X, Y, \dots , and Z all equal 1,

$$\overline{1 \cdot 1 \cdot \dots \cdot 1} = \bar{1} + \bar{1} + \dots + \bar{1} \\ \bar{1} = \bar{1} \\ 0 = 0$$

If X, Y, \dots , and Z do not all equal 1, then one or more of these literals must equal 0. If even one of the literals equals 0,

$$\overline{0 \cdot 1 \cdot \dots \cdot 1} = \bar{0} + \bar{1} + \dots + \bar{1} \\ \bar{0} = 1 + 0 + \dots + 0 \\ 1 = 1$$

Theorem 7a states that a product of literals may be complemented by changing the product to a sum of the literals and complementing each literal. Theorem 7b states that a sum of literals may be complemented by changing the sum to a product of the literals and complementing each literal.

EXAMPLES:

$$\overline{ABC\bar{D}E} = A + \bar{B} + \bar{C} + D + \bar{E}$$

$$\overline{A + \bar{B} + \bar{C} + D + \bar{E}} = \bar{A}BC\bar{D}E$$

Note that Theorems 7a and 7b and the above examples represent equivalences. For example, $\overline{ABC\bar{D}E}$ is *equivalent* to $A + \bar{B} + \bar{C} + D + \bar{E}$, but $\bar{A}BC\bar{D}E$ and $A + \bar{B} + \bar{C} + D + \bar{E}$ are *complements* of each other.

This pair of theorems may be written in a more general form as in Theorem 8.

$$8. \quad \bar{f}(X, Y, \dots, Z, \cdot, +) = f(\bar{X}, \bar{Y}, \dots, \bar{Z}, +, \cdot)$$

This theorem is read as follows. Given an expression containing literals such as X, Y , and Z , and occurrences of the operators \cdot and $+$. To complement this expression, signified by the \bar{f} , each literal is complemented, each \cdot is changed to $+$, and each $+$ is changed to \cdot . A simple example follows.

$$\overline{C + A\bar{B}} = \bar{C}(\bar{A} + B)$$

Note the importance of the parentheses. In the original expression, C is added to the product $A\bar{B}$. Therefore, the complement \bar{C} must be multiplied by the sum $(\bar{A} + B)$. Now for a more complex example:

$$\overline{(A\bar{B} + C)\bar{D} + E} = [(\bar{A} + B)\bar{C} + D]\bar{E}$$

Again note the importance of parentheses and brackets. The product $A\bar{B}$, when complemented, becomes $(\bar{A} + B)$. This product was originally added to C ; therefore, the sum $(\bar{A} + B)$ is now multiplied by \bar{C} . The sum $(A\bar{B} + C)$ was originally multiplied by \bar{D} ; therefore, the product $(\bar{A} + B)\bar{C}$ is now added to D . Finally, E was originally added to $(A\bar{B} + C)\bar{D}$; therefore, \bar{E} is now multiplied by $(\bar{A} + B)\bar{C} + D$, giving $[(\bar{A} + B)\bar{C} + D]\bar{E}$.

Some important simplification theorems now follow.

$$9a. XY + XZ = X(Y + Z)$$

$$9b. (X + Y)(X + Z) = X + YZ$$

Theorem 9a is like factoring in ordinary algebra. The operation represented by Theorem 9b is not permitted in ordinary algebra, but the procedure is analogous to that in Theorem 9a. The procedure may be better understood if Theorem 9a is first considered in a little different way. The X is common to both terms XY and XZ . X is *multiplied* by Y , and X is *multiplied* by Z . Therefore, X will be *multiplied* by the *sum* of the remainders of each term, namely $(Y + Z)$, giving $X(Y + Z)$.

Now, Theorem 9b can be thought of in a similar way. X is common to both factors $(X + Y)$ and $(X + Z)$. X is *added* to Y , and X is *added* to Z . Therefore, X will be *added* to the *product* of the remainders of each factor, namely YZ , giving $X + YZ$.

EXAMPLES:

$$(a) \quad AB + A\bar{C}D + A(E + \bar{F}) = A(B + \bar{C}D + E + \bar{F})$$

$$(b) \quad (A + B)(A + \bar{C} + D)(A + E\bar{F}) = A + B(\bar{C} + D)E\bar{F}$$

The examples have purposely been presented in dual pairs so that the similarity of the dual operations can be more easily seen. This practice will be maintained throughout the study of theorems.

In example (a), A is common to all three terms. In each case A is *multiplied* by the remainder of the term. Therefore, the A will be *multiplied* by the *sum* of what remains in each term: A is multiplied by the sum $(B + \bar{C}D + E + \bar{F})$. $E + \bar{F}$ does not require additional parentheses (see Theorem 6).

In example (b), A is common to every factor. In each case, A is *added* to the remainder of the factor. Therefore, the A will be *added* to the *product* of what remains in each factor: A is added to $B(\bar{C} + D)E\bar{F}$. Again, $E\bar{F}$ does not require additional parentheses (see Theorem 6).

Now for two slightly more involved examples:

$$(a) \quad A\bar{B}CD + A\bar{B}CE + ACF = AC(\bar{B}D + \bar{B}E + F) \\ = AC[\bar{B}(D + E) + F]$$

$$(b) \quad (A + \bar{B} + C + D)(A + \bar{B} + C + E)(A + C + F) \\ = A + C + (\bar{B} + D)(\bar{B} + E)F \\ = A + C + (\bar{B} + DE)F$$

In example (a), AC is common to all three terms and is *multiplied* by the remainder of a term in each case. Therefore, AC is *multiplied* by the *sum* of the remainder of each term, namely $(\bar{B}D + \bar{B}E + F)$. Furthermore, within the parentheses, \bar{B} is common to two terms; therefore, \bar{B} is similarly factored out to obtain the final expression.

In example (b), $A + C$ is common to all three factors and is *added* to the remainder of the factor in each case. Therefore, $A + C$ is *added* to the *product* of the remainder of each factor, namely $(\bar{B} + D)(\bar{B} + E)F$. Furthermore, \bar{B} is common to two of the factors, leading to the final expression.

$$10a. \quad XY + X\bar{Y} = X$$

$$10b. \quad (X + Y)(X + \bar{Y}) = X$$

This theorem may appear to be a special case of Theorem 9. In Theorem 10a,

$$XY + X\bar{Y} = X(Y + \bar{Y}) \\ = X \cdot 1 = X$$

In Theorem 10b,

$$(X + Y)(X + \bar{Y}) = X + Y\bar{Y} \\ = X + 0 = X$$

However, this theorem has further implications. In a sum of 2^m n -variable terms, or in a product of 2^m n -variable factors, if m variables occur in all possible combinations (represented by Y and \bar{Y} in the theorem), while the remaining $n - m$ variables are constant (represented by X in the theorem), the m variables are redundant and the $n - m$ variables define the expression.

An example with $n = 3$ and $m = 2$ is as follows:

$$X\bar{Y}\bar{Z} + X\bar{Y}Z + XY\bar{Z} + XYZ = X \cdot 1 = X$$

Here, in $2^m = 2^2 = 4$ terms, $m = 2$ variables (Y and Z) occur in all possible combinations, while $n - m = 1$ variable (X) is constant; thus, Y and Z are redundant and X defines the expression.

Another example with $n = 4$ and $m = 2$ is as follows:

$$(\bar{W} + X + \bar{Y} + \bar{Z})(\bar{W} + X + \bar{Y} + Z) \\ (\bar{W} + X + Y + \bar{Z})(\bar{W} + X + Y + Z) = \bar{W} + X + 0 = \bar{W} + X$$

Y and Z occur in all possible combinations, whereas \bar{W} and X are constant; thus the expression reduces to $\bar{W} + X$.

Note that the number of terms or factors involved in such a simplification must be a power of two (2, 4, 8, 16, etc.), since there are 2^m combinations of m variables.

This theorem is the basis of other simplification methods which will be taken up in Chapters 6 and 7.

$$11a. X + XY = X$$

$$11b. X(X + Y) = X$$

Theorem 11a can be proved as follows:

$$X + XY = X(1 + Y) = X \cdot 1 = X$$

Although Theorem 11b can be considered proved once Theorem 11a is proved, since the theorems are duals of each other, Theorem 11b can also be proved as follows:

$$X(X + Y) = XX + XY = X + XY$$

and the rest of the steps will be the same as in the proof of Theorem 11a. Another proof is as follows:

$$X(X + Y) = (X + 0)(X + Y) = X + 0 \cdot Y = X + 0 = X$$

This simplification theorem may be applied in the following way. If a smaller term (or factor) appears in a larger term (or factor), then the larger term (or factor) is redundant.² In Theorem 11a, X appears in the larger term XY . Therefore, the term XY is redundant and the expression reduces to X . Similarly, in Theorem 11b, X appears in the larger factor $(X + Y)$; therefore, the factor $(X + Y)$ is redundant and the expression reduces to X .

EXAMPLES:

$$(a) \quad A\bar{B} + A\bar{B}C + A\bar{B}(D + E) = A\bar{B}$$

$$(b) \quad (A + \bar{B})(A + \bar{B} + C)(A + \bar{B} + DE) = A + \bar{B}$$

In example (a), the first term $A\bar{B}$ appears in the second and third terms; therefore, the second and third terms are redundant and the expression reduces to $A\bar{B}$. In example (b), the first factor $(A + \bar{B})$ appears in the second and third factors; the second and third factors are therefore redundant and this expression reduces to $A + \bar{B}$.

$$12a. X + \bar{X}Y = X + Y$$

$$12b. X(\bar{X} + Y) = XY$$

A few interesting ways to prove Theorem 12a follow:

²The smaller term (or factor) is defined as the one containing fewer literals; conversely, the larger term (or factor) is the one containing more literals. The larger is said to *subsume* the smaller.

$$\begin{aligned}
 X + \bar{X}Y &= (X + \bar{X})(X + Y) && \text{(Theorem 9b in reverse)} \\
 &= 1 \cdot (X + Y) \\
 &= X + Y
 \end{aligned}$$

or

$$\begin{aligned}
 X + \bar{X}Y &= X + XY + \bar{X}Y && \text{(Theorem 11a in reverse)} \\
 &= X + Y && \text{(Theorem 10a)}
 \end{aligned}$$

Theorem 12b can be proved in the same manner.

Proofs of the type just shown represent interesting manipulations of the algebra. However, a straightforward method of proof that can always be used is called the "truth table" proof, which is a means of applying the method of perfect induction. In a proof by perfect induction, it is shown that an equality of expressions exists for all possible combinations of values of the variables. This type of proof is especially adaptable to Boolean algebra where the variables can have only two values, 0 or 1. A truth table proof of Theorem 12a follows:

1	2	3	4	5	6
X	Y	\bar{X}	$\bar{X}Y$	$X + \bar{X}Y$	$X + Y$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

First, every possible combination of the values of the variables are listed. In this case, with two variables, X and Y , there are four possible combinations, 00, 01, 10, and 11. These combinations are listed in columns 1 and 2. Since an \bar{X} will be needed, the complementary values of column 1 are written in column 3. Next, the product $\bar{X}Y$ is required; this is placed in column 4, and is obtained by the multiplication of columns 2 and 3. In column 5 is written the values of the sum $X + \bar{X}Y$, which is obtained by the addition of columns 1 and 4. Finally, in column 6, is written the sum $X + Y$, which is obtained by the addition of columns 1 and 2. It is now found that the values in columns 5 and 6 agree for every possible combination of the variables X and Y , thus proving the theorem.

In practice it is found helpful to think about Theorem 12 in a slightly more general way, as shown in Theorem 12'.

$$12a' \quad ZX + Z\bar{X}Y = ZX + ZY$$

$$12b' \quad (Z + X)(Z + \bar{X} + Y) = (Z + X)(Z + Y)$$

The reason for presenting this modification of Theorem 12 is that the

application is usually encountered in the form of Theorem 12'. The way of applying this theorem is as follows: if a smaller term (or factor) appears in a larger term (or factor) except that *one* variable in the smaller term (or factor) and the corresponding variable in the larger term (or factor) are complements, then that variable in the *larger* term (or factor) is redundant.

In Theorem 12a', the smaller term ZX , appears in the larger term $Z\bar{X}Y$, except for the complementary X and \bar{X} . Therefore, the \bar{X} in the larger term is redundant, and the expression reduces to $ZX + ZY$. A similar relationship exists in Theorem 12b'. It does not matter where the "bar" actually appears; it is always the variable in the *larger* term of factor that is redundant.

EXAMPLE:

$$W\bar{X} + WXY = W\bar{X} + WY$$

The X in the larger term is redundant because it is the complement of the \bar{X} in the smaller term.

Note the difference between the application of Theorems 11 and 12'. Theorem 11: if a smaller term (or factor) appears exactly "as is" in a larger term (or factor), *the entire larger term (or factor)* is redundant. Theorem 12': if a smaller term (or factor) appears in a larger term (or factor) except for one complemented variable, *only that variable in the larger term (or factor)* is redundant. If a smaller term (or factor) appears in a larger term (or factor) with two or more variables complemented, no simplification of this sort is possible.

Now for some examples involving Theorems 11 and 12'.

- (a) $A\bar{B} + A\bar{B}C + \bar{A}\bar{B}D + ABE + \bar{A}BF = A\bar{B} + \bar{B}D + AE + \bar{A}BF$
 (b) $(A + \bar{B})(A + \bar{B} + C)(\bar{A} + \bar{B} + D)(A + B + E)(\bar{A} + B + F)$
 $= (A + \bar{B})(\bar{B} + D)(A + E)(\bar{A} + B + F)$

In example (a), the first term appears in the second with no complementation; therefore, the entire second term is redundant. In the third and fourth terms, one variable appears in complemented form: \bar{A} in the third, and B in the fourth; these two variables are therefore redundant. In the fifth term, two variables are complemented and therefore no simplification of this sort is possible. The final expression may be factored in one of two possible ways if desired. Example (b) is the dual of example (a) and the same reasoning can be made throughout.

The following pair of theorems can be thought of as the "included term" and "included factor" theorems, respectively.

$$13a. XY + \bar{X}Z + YZ = XY + \bar{X}Z$$

$$13b. (X + Y)(\bar{X} + Z)(Y + Z) = (X + Y)(\bar{X} + Z)$$

An interesting proof of Theorem 13a is as follows.

$$\begin{aligned}
 XY + \bar{X}Z + YZ &= XY + \bar{X}Z + YZ(X + \bar{X}) \\
 &= XY + \bar{X}Z + XYZ + \bar{X}YZ \\
 &= XY + \bar{X}Z
 \end{aligned}$$

Theorem 13b may be proved in a similar manner.

Following is a truth table proof of Theorem 13a.

X	Y	Z	\bar{X}	XY	$\bar{X}Z$	YZ	$XY + \bar{X}Z$	$XY + \bar{X}Z + YZ$
0	0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	1	1
0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	1	1
1	1	1	0	1	0	1	1	1

The last two columns have the same value for all possible combinations of values of the variables, proving the equivalence. However, more can be learned from this truth table. Examination of the YZ column and the $XY + \bar{X}Z$ column will show that $XY + \bar{X}Z$ equals 1 for four of the eight possible combinations, whereas YZ equals 1 for two of the eight possible combinations. Furthermore, the two combinations for which YZ equals 1 are included among the four combinations for which $XY + \bar{X}Z$ equals 1; that is, the expression $XY + \bar{X}Z$ "includes" the term YZ . Hence the name included term (and included factor) theorem.

Now for the recognition of the application of this pair of theorems. In the application of Theorem 13a, two terms are looked for: one that contains a variable, and the other that contains the complement of this same variable. For instance, the first term in the theorem contains an X and the second term contains an \bar{X} . If two such terms are found, the remainders of each term, exclusive of this variable and its complement, together form a product that is included by the first two terms. In Theorem 13a, the first term contains an X and the second term contains an \bar{X} . The remainders of these two terms are Y and Z , respectively, and together they form a product YZ which is included by the first two terms. An included term may lead to the elimination of redundancy in the expression. For example, in the theorem, the third term YZ is redundant: there is no need to add the term YZ to the terms $XY + \bar{X}Z$, since the term YZ is already included.

In a sense, Theorem 13a is first applied in reverse to obtain the included term. The included term may then be used to eliminate redundancy in the

expression, following which, Theorem 13a is applied to eliminate the included term.

Similar reasoning applies in Theorem 13b. Two factors are looked for: one that contains a variable, and the other that contains the complement of this variable. If two such factors are found, the remainders of each factor, exclusive of this variable and its complement, together form a sum that is included by the first two factors. An included factor may be used to eliminate redundancy in the expression.

Theorem 13 can often be used in conjunction with other theorems, such as Theorems 10, 11, and 12'. Some examples follow:

$$(a) \quad AB + \bar{A}C + BCD = AB + \bar{A}C$$

The first two terms, noting the A and \bar{A} , include a term BC . Because of the included term BC , the term BCD is redundant (Theorem 11a). Therefore the expression reduces to $AB + \bar{A}C$.

$$(b) \quad (A + B)(\bar{A} + C)(B + C + D) = (A + B)(\bar{A} + C)$$

The first two factors, again noting the A and \bar{A} , include a factor $(B + C)$. $(B + C)$ appears in the third factor $(B + C + D)$; therefore, the factor $(B + C + D)$ is redundant (Theorem 11b).

$$(c) \quad AB + \bar{A}C + \bar{B}CD = AB + \bar{A}C + CD \\ = AB + C(\bar{A} + D)$$

The first two terms include the term BC . BC appears in the third term $\bar{B}CD$ except that the \bar{B} in the third term is complemented. Therefore, the \bar{B} is redundant (Theorem 12a') and the expression reduces to $AB + \bar{A}C + CD$ (which may be factored).

The next example is the dual of example (c).

$$(d) \quad (A + B)(\bar{A} + C)(\bar{B} + C + D) = (A + B)(\bar{A} + C)(C + D) \\ = (A + B)(C + \bar{A}D)$$

$$(e) \quad AB + \bar{A}C + \bar{B}C = AB + C$$

The first two terms include BC . BC and $\bar{B}C$ reduce to C (Theorem 10a). The expression at this point reads $AB + \bar{A}C + C$. The $\bar{A}C$ term is redundant because of the C term. Therefore, the expression reduces to $AB + C$.

The next example is the dual of example (e).

$$(f) \quad (A + B)(\bar{A} + C)(\bar{B} + C) = (A + B)C$$

$$(g) \quad ABC + \bar{A}BD + BCDE = ABC + \bar{A}BD$$

The first two terms include BCD , which makes $BCDE$ redundant.

Included terms may lead to other included terms. For example,

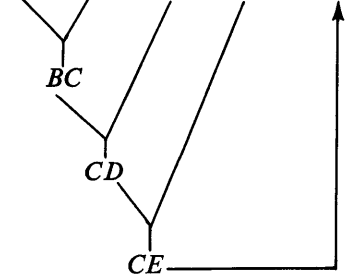
$$(h) \quad AB + \bar{A}C + \bar{B}D + CD = AB + \bar{A}C + \bar{B}D$$

The first two terms include BC . BC and $\bar{B}D$ include CD . The fourth term CD therefore is redundant, and the expression reduces to $AB + \bar{A}C + \bar{B}D$.

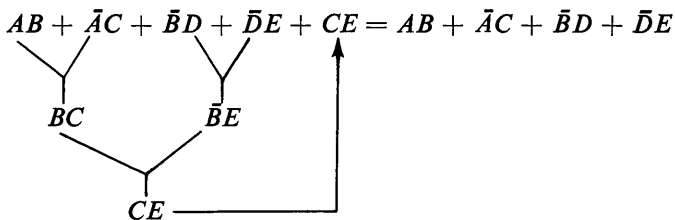
Additional examples are given for analysis.

$$(i) \quad (A + B)(\bar{A} + C)(\bar{B} + D)(C + D) = (A + B)(\bar{A} + C)(\bar{B} + D)$$

$$(j) \quad AB + \bar{A}C + \bar{B}D + \bar{D}E + CE = AB + \bar{A}C + \bar{B}D + \bar{D}E$$



or



$$(k) \quad (A + B)(\bar{A} + C)(\bar{B} + D)(\bar{D} + E)(C + E) \\ = (A + B)(\bar{A} + C)(\bar{B} + D)(\bar{D} + E)$$

$$(l) \quad AB + \bar{A}C + A\bar{C} = \bar{A}C + A\bar{C} + BC$$

If the term AB is eliminated, the term BC cannot be eliminated since it would no longer be included by $AB + \bar{A}C$.

Resumé of Simplification Theorems and "Method of Attack"

While no hard and fast rules can be given for the best "method of attack" in simplifying any Boolean expression, the following approach is given as a guide.

Resumé of Simplification Theorems

1a. $0 \cdot X = 0$

1b. $1 + X = 1$

2a. $1 \cdot X = X$

2b. $0 + X = X$

3a. $XX = X$

3b. $X + X = X$

4a. $X\bar{X} = 0$

4b. $X + \bar{X} = 1$

...

9a. $XY + XZ = X(Y + Z)$

9b. $(X + Y)(X + Z) = X + YZ$

10a. $XY + X\bar{Y} = X$

10b. $(X + Y)(X + \bar{Y}) = X$

11a. $X + XY = X$

11b. $X(X + Y) = X$

12a. $X + \bar{X}Y = X + Y$

12b. $X(\bar{X} + Y) = XY$

12a'. $ZX + Z\bar{X}Y = ZX + ZY$

12b'. $(Z + X)(Z + \bar{X} + Y) = (Z + X)(Z + Y)$

13a. $XY + \bar{X}Z + YZ = XY + \bar{X}Z$

13b. $(X + Y)(\bar{X} + Z)(Y + Z) = (X + Y)(\bar{X} + Z)$

Theorems 1 to 4 are, of course, applied whenever possible; however, their application becomes almost "second nature," and more deliberate thought is usually directed toward the other less obvious theorems. It is stressed again that X may represent not only a variable but also a term or factor or more complex expression.

Theorems 10, 11, and 12' should be applied exhaustively. Then Theorem 13 should be applied. Theorems 10 through 12' may further be applied in conjunction with or following the application of Theorem 13.

If a "factored" form, rather than a sum of products or product of sums form, is desired, then Theorem 9 may be applied. Theorem 9 generally should not be applied until there is no longer any possible application of the other theorems; if Theorem 9 is applied too early, the application of the other theorems may be obscured.

There is a tendency for the beginner to prefer to work with the sum of products form, rather than the product of sums form. To this end he may (1) "multiply out" a product of sums expression (that is, apply Theorem 9a in reverse) to obtain an equivalent sum of products expression or; (2) complement the product of sums expression to obtain a complementary sum of products, and after simplification, re complement or; (3) obtain the dual of the product of sums expression and, after simplifying, obtain the dual of the dual.

This is an undesirable practice! Every additional operation adds a potential source of error. Also, multiplying out generally adds additional redun-

dancy which must be removed. There is no need for this practice: each sum of products theorem has its dual product of sums theorem, and both can be used with equal facility.

Additional Theorems

The pair of theorems that follow are not simplification theorems but rather transposition theorems.

$$14a. XY + \bar{X}Z = (X + Z)(\bar{X} + Y)$$

$$14b. (X + Y)(\bar{X} + Z) = XZ + \bar{X}Y$$

The key point to look for in the possibility of making a transposition of this type is two terms or factors, one that contains a variable, and the other that contains the complement of this same variable. In Theorem 14a, the first term contains an X and the second term contains an \bar{X} ; therefore, the transposition shown can be made by adding the X to the remainder of the term containing the \bar{X} , and adding the \bar{X} to the remainder of the term containing the X , these two sums being multiplied together. Conversely, in Theorem 14b there are two factors, one that contains an X , and the other that contains an \bar{X} . The transposition shown can be made by multiplying the X by the remainder of the factor containing the \bar{X} , and multiplying the \bar{X} by the remainder of the factor containing the X , and then adding these two products together.

EXAMPLE:

$$A\bar{B}C + \bar{A}(\bar{D} + E) = (A + \bar{D} + E)(\bar{A} + \bar{B}C)$$

In making the transposition, the A is added to the remainder of the term containing the \bar{A} , namely $\bar{D} + E$, and the \bar{A} is added to the remainder of the term containing the A , namely $\bar{B}C$, the two sums being multiplied together.

For a second example, the transposition will be made in the other direction, starting with $(A + \bar{D} + E)(\bar{A} + \bar{B}C)$. Here the A is multiplied by the remainder of the factor containing the \bar{A} , namely $\bar{B}C$, and the \bar{A} is multiplied by the remainder of the factor containing the A , namely $\bar{D} + E$, the two products being added together. The result is the original expression $A\bar{B}C + \bar{A}(\bar{D} + E)$.

When the relationship between Boolean algebra and switching circuits is taken up later, the desirability of such transpositions will be apparent. Two special cases of the application of this theorem are given because of the frequency with which they are encountered in practice.

$$\begin{aligned}X\bar{Y} + \bar{X}Y &= (X + Y)(\bar{X} + \bar{Y}) \\ \bar{X}\bar{Y} + XY &= (\bar{X} + Y)(X + \bar{Y})\end{aligned}$$

The first case describes the “exclusive or” function (one or the other but not both), sometimes symbolized by $X \oplus Y$ or $X \vee Y$. Another useful form of the expression is $(X + Y)\bar{X}\bar{Y}$. The second case describes the complement of the “exclusive or”: the “neither or both” or “if and only if” function, sometimes symbolized by $X \equiv Y$. Another useful form of the expression is $(\bar{X} + \bar{Y}) + XY$.

$$\mathbf{15a.} \quad X \cdot f(X, \bar{X}, Y, \dots, Z) = X \cdot f(1, 0, Y, \dots, Z)$$

$$\mathbf{15b.} \quad X + f(X, \bar{X}, Y, \dots, Z) = X + f(0, 1, Y, \dots, Z)$$

Theorem 15a states that if a variable X is multiplied by an expression containing occurrences of X or \bar{X} , then all X 's in the expression may be replaced by 1's, and all \bar{X} 's in the expression may be replaced by 0's. This can be seen to be permissible since

$$X \cdot X = X \cdot 1 = X$$

and

$$X \cdot \bar{X} = X \cdot 0 = 0$$

Theorem 15b states that if a variable X is added to an expression containing occurrences of X and \bar{X} , then all X 's in the expression may be replaced by 0's, and all \bar{X} 's may be replaced by 1's. Again this is permissible since

$$X + X = X + 0 = X$$

and

$$X + \bar{X} = X + 1 = 1$$

EXAMPLES:

$$\begin{aligned}\mathbf{(a)} \quad A \cdot [AB + \bar{A}C + (A + D)(\bar{A} + E)] &= A \cdot [1 \cdot B + 0 \cdot C + (1 + D)(0 + E)] \\ &= A[B + 0 + 1 \cdot E] \\ &= A(B + E)\end{aligned}$$

$$\begin{aligned}\mathbf{(b)} \quad \bar{A} \cdot (AB + \bar{A}C + D) &= \bar{A} \cdot (0 \cdot B + 1 \cdot C + D) \\ &= \bar{A}(C + D)\end{aligned}$$

Not only is this theorem useful for simplification, but it is partly the basis of the following theorem.

$$\mathbf{16a.} \quad f(X, \bar{X}, Y, \dots, Z) = X \cdot f(1, 0, Y, \dots, Z) + \bar{X} \cdot f(0, 1, Y, \dots, Z)$$

$$\mathbf{16b.} \quad f(X, \bar{X}, Y, \dots, Z) = [X + f(0, 1, Y, \dots, Z)][\bar{X} + f(1, 0, Y, \dots, Z)]$$

This pair of theorems can be proved using Theorem 10 in reverse, along

with Theorem 15. In Theorem 16a an expression is multiplied first by X and also by \bar{X} , the two products being added together.

$$f(X, \bar{X}, Y, \dots, Z) = X \cdot f(X, \bar{X}, Y, \dots, Z) + \bar{X} \cdot f(X, \bar{X}, Y, \dots, Z)$$

It can be seen that these two expressions are equivalent since the latter can be reduced to the former by the application of Theorem 10.

Now, by the application of Theorem 15 the X 's and \bar{X} 's can be replaced with 1's and 0's, respectively, when the expression is multiplied by X , and they can be replaced by 0's and 1's, respectively, when the expression is multiplied by \bar{X} . Theorem 16b can be similarly proved.

Theorem 16 has the following application: given an expression containing any number of occurrences of some variable and its complement, say X and \bar{X} , the expression can be rewritten using only one occurrence of X and one occurrence of \bar{X} , at most.

EXAMPLE:

$$AB + \bar{A}C + (A + D)E + (\bar{A} + F)G$$

Find an equivalent expression with only one occurrence of A and one occurrence of \bar{A} , at most.

$$\begin{aligned} & AB + \bar{A}C + (A + D)E + (\bar{A} + F)G \\ = & A[AB + \bar{A}C + (A + D)E + (\bar{A} + F)G] \\ & + \bar{A}[AB + \bar{A}C + (A + D)E + (\bar{A} + F)G] \\ = & A[1 \cdot B + 0 \cdot C + (1 + D)E + (0 + F)G] \\ & + \bar{A}[0 \cdot B + 1 \cdot C + (0 + D)E + (1 + F)G] \\ = & A[B + 0 + 1 \cdot E + FG] + \bar{A}[0 + C + DE + 1 \cdot G] \\ = & A[B + E + FG] + \bar{A}[C + DE + G] \end{aligned}$$

The application of Theorem 16b is analogous.

While this pair of theorems reduces one variable to one occurrence of itself and its complement at most, it may introduce multiplicity of other variables. Circuit requirements, however, may make this a desirable operation.

Theorem 16 may be further applied to each bracketed expression independently, thereby reducing a selected second variable to two occurrences of itself and its complement, at most. A third selected variable may be reduced to four occurrences of itself and its complement, etc.

Boolean algebra will now be applied to the simplification of the XYZ Insurance Company Manual statement at the beginning of this chapter.

Let

- A = Applicant has been issued Policy No. 19
- B = Applicant is married
- C = Applicant is a male
- D = Applicant is under 25

Policy No. 22 may be issued only if

- 1. ABC
- or 2. ABD
- or 3. $\bar{A}\bar{B}\bar{C}$
- or 4. CD
- or 5. $B\bar{D}$

which can be written

$$\begin{aligned}
 & ABC + ABD + \bar{A}\bar{B}\bar{C} + CD + B\bar{D} \\
 = & ABC + AB + \bar{A}\bar{B}\bar{C} + CD + B\bar{D} && \text{(Theorem 12a')} \\
 = & AB + B\bar{C} + CD + B\bar{D} && \text{(Theorems 11a, 12a')} \\
 = & AB + B + CD + B\bar{D} && \text{(Theorems 13a, 10a)} \\
 = & B + CD && \text{(Theorem 11a)}
 \end{aligned}$$

Policy No. 22 may be issued only if the applicant

- 1. Is married,
- or 2. Is a male under 25.

Summary of Boolean Algebra Postulates and Theorems

Postulates

$X = 1$ or else $X = 0$

$$\begin{array}{ll}
 1 \cdot 1 = 1 & 0 + 0 = 0 \\
 1 \cdot 0 = 0 \cdot 1 = 0 & 0 + 1 = 1 + 0 = 1 \\
 0 \cdot 0 = 0 & 1 + 1 = 1 \\
 \bar{1} = 0 & \bar{0} = 1
 \end{array}$$

Theorems

$$\begin{array}{ll}
 1a. 0 \cdot X = 0 & 1b. 1 + X = 1 \\
 2a. 1 \cdot X = X & 2b. 0 + X = X \\
 3a. XX = X & 3b. X + X = X \\
 4a. X\bar{X} = 0 & 4b. X + \bar{X} = 1 \\
 5a. XY = YX & 5b. X + Y = Y + X \\
 6a. XYZ = (XY)Z = X(YZ) & 6b. X + Y + Z = (X + Y) + Z \\
 & \quad = X + (Y + Z) \\
 7a. \overline{XY \dots Z} = \bar{X} + \bar{Y} + \dots + \bar{Z} & 7b. \overline{X + Y + \dots + Z} = \bar{X}\bar{Y} \dots \bar{Z} \\
 8. \bar{f}(X, Y, \dots, Z, \cdot, +) = f(\bar{X}, \bar{Y}, \dots, \bar{Z}, +, \cdot) & \\
 9a. XY + XZ = X(Y + Z) & 9b. (X + Y)(X + Z) = X + YZ
 \end{array}$$

10a. $XY + X\bar{Y} = X$

10b. $(X + Y)(X + \bar{Y}) = X$

11a. $X + XY = X$

11b. $X(X + Y) = X$

12a. $X + \bar{X}Y = X + Y$

12b. $X(\bar{X} + Y) = XY$

12a'. $ZX + Z\bar{X}Y = ZX + ZY$

12b'. $(Z + X)(Z + \bar{X} + Y) = (Z + X)(Z + Y)$

13a. $XY + \bar{X}Z + YZ = XY + \bar{X}Z$

13b. $(X + Y)(\bar{X} + Z)(Y + Z) = (X + Y)(\bar{X} + Z)$

14a. $XY + \bar{X}Z = (X + Z)(\bar{X} + Y)$

14b. $(X + Y)(\bar{X} + Z) = XZ + \bar{X}Y$

15a. $X \cdot f(X, \bar{X}, Y, \dots, Z) = X \cdot f(1, 0, Y, \dots, Z)$

15b. $X + f(X, \bar{X}, Y, \dots, Z) = X + f(0, 1, Y, \dots, Z)$

16a. $f(X, \bar{X}, Y, \dots, Z) = X \cdot f(1, 0, Y, \dots, Z) + \bar{X} \cdot f(0, 1, Y, \dots, Z)$

16b. $f(X, \bar{X}, Y, \dots, Z) = [X + f(0, 1, Y, \dots, Z)][\bar{X} + f(1, 0, Y, \dots, Z)]$

PROBLEMS

1. Simplify:

(a) $A + \bar{B} + \bar{A}B + (A + \bar{B})\bar{A}B$

(b) $(A + \bar{B} + \bar{A}B)(A + \bar{B})\bar{A}B$

(c) $A + \bar{B} + \bar{A}B + \bar{C}$

(d) $(A + \bar{B} + \bar{A}B)\bar{C}$

(e) $(A + \bar{B})\bar{A}B + \bar{C}$

(f) $(A + \bar{B})\bar{A}B\bar{C}$

Hint: $A + \bar{B}$ and $\bar{A}B$ are complements.

2. Complement:

(a) $[(\bar{A}B + \bar{C})D + \bar{E}]F$

(b) $S[\bar{W} + I(T + \bar{C})] + H$

*(c) $F[\bar{R}(I + \bar{D}A) + \bar{Y}]$

*(d) $U + [(V + \bar{W})X + \bar{Y}]Z$

3. Reduce to minimum number of literals.

(a) $CD(E + A)F + (A + E)BC$

(b) $(A + BF + C + E)(D + E + FB)$

(c) $ABC(D + E) + F(E + D)(G + H)B$

(d) $(A + CE + B + F)(D + GH + F + EC)$

(e) $AB(C + D)(\bar{E} + F) + \bar{G}(D + C)\bar{H}A$

(f) $(A\bar{B} + \bar{C} + D + EF)(\bar{G} + FE + \bar{C} + \bar{H}K)$

*(g) $BC(\bar{D} + F)(G + \bar{H}) + J(F + \bar{D})\bar{K}B$

*(h) $(L + \bar{M} + N\bar{P} + \bar{Q}R)(R\bar{Q} + S + \bar{M} + T)$

*(i) $\bar{A}(B + C)\bar{D}E + EF(C + B)(\bar{G} + H)$

$$*(j) (F\bar{R} + I + D + AY)(YA + \bar{Q} + U + I)$$

4. Simplify:

$$(a) A\bar{B}\bar{C} + A\bar{B}\bar{C}D + \bar{C}A$$

$$(b) A\bar{B}C + \bar{A}\bar{C}D + \bar{C}A$$

$$(c) (A + B + CD)(\bar{A} + B)(\bar{A} + B + E)$$

$$(d) DEH + \bar{E}G\bar{H} + \bar{H}E + HF\bar{E} + J\bar{H}E$$

$$(e) (K + L + \bar{P})(L + M + P)(Q + P + \bar{L})(L + \bar{P})(\bar{P} + N + \bar{L})$$

$$(f) (A + BC)(A + \bar{B} + \bar{C} + D)(\bar{A} + BC + E)(\bar{A} + \bar{B} + \bar{C} + F) \\ (A + BC + G)$$

$$*(g) IBM + K\bar{I}M + \bar{M}I + \bar{M}\bar{I}G + N\bar{I}\bar{M}$$

$$*(h) (I + C + B + \bar{M})(\bar{I} + M)(M + \bar{I} + L)(\bar{M} + \bar{I} + X)(V + I + M)$$

5. Simplify:

$$(a) AB + \bar{A}C\bar{D}E + \bar{B}C\bar{D}$$

$$(b) ABC\bar{D} + B\bar{C}\bar{E} + AE$$

$$(c) (A + B)(\bar{A} + C + \bar{D})(\bar{B} + C + \bar{D})$$

$$(d) (A + B + \bar{C})(B + \bar{C} + \bar{D})(A + D)$$

$$(e) \bar{A}B\bar{C} + AD + B\bar{C}\bar{D}$$

$$(f) \bar{A}B\bar{C} + \bar{A}BD + CD$$

$$(g) (\bar{A} + B + \bar{C})(C + D)(\bar{A} + B + \bar{D} + E)$$

$$(h) (\bar{A} + B + \bar{C})(\bar{A} + B + D + E)(C + D)$$

6. Simplify:

$$(a) AC + \bar{B}\bar{A} + \bar{D}\bar{C}\bar{B} + \bar{C}EB + \bar{B}CF + B\bar{G}C$$

$$(b) (P + A + T)(P + E + \bar{T})(\bar{P} + O + T)(\bar{P} + U + \bar{T})(P + I)(\bar{I} + \bar{T})$$

$$(c) \bar{A}BC + CE + BCD + \bar{D}\bar{E}$$

$$(d) (\bar{A} + B)(C + A)(B + \bar{C} + D)(B + D + E)$$

$$(e) ABD + ACD + A\bar{B}E + AF + \bar{E}\bar{F}$$

$$(f) \bar{K}L + \bar{L}M + HKM + \bar{G}\bar{M} + \bar{G}HJ$$

$$(g) \bar{X}ZY + \bar{Y}\bar{X}Z + \bar{Y}ZX + \bar{Z}YX + X\bar{Y}$$

$$(h) (\bar{A} + B)(A + \bar{C} + \bar{B})(B + A + \bar{C})(B + \bar{C} + \bar{A})(C + \bar{B} + \bar{A})$$

$$*(i) (B + A + D)(B + E + \bar{D})(\bar{B} + I + D)(\bar{B} + U + \bar{D}) \\ (\bar{B} + \bar{O})(O + D)$$

$$*(j) IOU + \bar{U}E + AIO + \bar{E}O$$

$$*(k) ABC + DE + ACF + A\bar{D} + A\bar{B}\bar{E}$$

$$*(l) (\bar{X} + Y)(X + \bar{Z} + \bar{Y})(Y + X + \bar{Z})(Y + \bar{Z} + \bar{X})(Z + \bar{Y} + \bar{X})$$

7. Transpose to a product of two expressions:

$$(a) A(B + \bar{C}) + \bar{A}\bar{D}E$$

$$(b) (A + \bar{B}\bar{C})\bar{D} + DE(F + G)$$

$$*(c) (A + \bar{B})CD + \bar{C}(EF + \bar{G})$$

$$*(d) \bar{T}\bar{U}(\bar{V} + \bar{W}) + (XY + Z)U$$

8. Transpose to a sum of two expressions:

(a) $(A + BC)(\bar{A} + \bar{D} + \bar{E})$

(b) $[A\bar{B} + \bar{C} + \bar{D}][D + (E + F)G]$

* (c) $[(A + \bar{B})C + D][F\bar{G} + \bar{D} + E]$

* (d) $[\bar{M}\bar{N} + \bar{O} + \bar{P}][(Q + R)S + O]$

9. Reduce the following to a single occurrence of A and \bar{A} . Express each as a product of two expressions and as a sum of two expressions.

(a) $AG + (A + B)C + \bar{A}D + (\bar{A} + F)E$

(b) $(A + \bar{B})(\bar{A} + C)(\bar{D} + E + AF)(G + \bar{H} + \bar{A}J)$

10. Find twelve ways to express with six literals, complementing variables only, i.e., without complementing products or sums.

$$\bar{A}C + \bar{A}B + A\bar{C} + A\bar{B}$$

2

Special Forms of Boolean Expressions

Four forms of Boolean expressions that are of particular interest are:

- Expanded sum of products
- Expanded product of sums
- “Minimum” sum of products
- “Minimum” product of sums

The expanded sum of products and expanded product of sums forms are useful for the analysis of Boolean expressions and their associated circuits and, also, they are a starting point for other methods of simplification which will be taken up later. The “minimum” sum of products and “minimum” product of sums forms are of interest because circuits are most frequently implemented directly from these expressions.

Expanded Sum of Products

In the expanded sum of products, each term contains every variable, either uncomplemented or complemented. To obtain the expanded sum of products from a sum of products, the missing variables are supplied in all possible combinations to each product. Actually, in so doing, Theorem 10a is used in reverse.

$$X = XY + X\bar{Y}$$

As an example, the sum of products

$$\bar{A}\bar{C}\bar{D} + A\bar{B}D + A\bar{C}$$

will be expanded. The first term, $\bar{A}\bar{C}\bar{D}$, has one missing variable B which is supplied in both its uncomplemented and complemented form; $\bar{A}\bar{C}\bar{D}$ thus expands into two terms: $\bar{A}\bar{B}\bar{C}\bar{D}$ and $\bar{A}B\bar{C}\bar{D}$. The term $A\bar{B}D$ also expands into two terms: $A\bar{B}\bar{C}D$ and $A\bar{B}CD$. The $A\bar{C}$ term has two missing variables B and D . Two variables can occur in four possible combinations. Therefore, the term $A\bar{C}$ expands into four terms: $A\bar{B}\bar{C}\bar{D}$, $A\bar{B}\bar{C}D$, $A\bar{B}C\bar{D}$, and $A\bar{B}CD$. The $A\bar{B}\bar{C}D$ term has already been obtained by the expansion of the $A\bar{B}D$ term and is not repeated. The expanded sum of products is therefore

$$(1) \quad \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}CD$$

Note, in this example, that the expanded sum of products contains seven of the sixteen possible combinations of the four variables. Although the “+” stands for the “inclusive or,” the nature of an expanded sum of products is such that all terms are mutually exclusive; that is, if one of the terms equals 1, all others must equal 0. For instance, if $A = 0$, $B = 0$, $C = 1$, and $D = 0$, the first expanded product, $\bar{A}\bar{B}\bar{C}\bar{D}$, is the only one equalling 1; all other expanded products will have one or more variables equalling 0, and thus all other products will equal 0. Of course it is possible, in an expanded sum of products, for all terms to equal 0.

Expanded Product of Sums

The expanded product of sums can be obtained from a product of sums in a similar manner. For example, the product of sums

$$(\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{C} + D)(A + C)(A + \bar{D})$$

expands into

$$(2) \quad (\bar{A} + \bar{B} + \bar{C} + \bar{D})(\bar{A} + \bar{B} + \bar{C} + D)(\bar{A} + B + \bar{C} + D)(A + \bar{B} + C + \bar{D})(A + \bar{B} + C + D)(A + B + C + \bar{D})(A + B + C + D)(A + \bar{B} + \bar{C} + \bar{D})(A + B + \bar{C} + \bar{D})$$

Here again, the missing variables are supplied in all possible combinations, this time to each sum.

The product of sums

$$(\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{C} + D)(A + C)(A + \bar{D})$$

above, is equivalent to the sum of products

$$\bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}D + A\bar{C}$$

used in the previous example. Equivalent expressions were purposely chosen for these examples to illustrate an important complementary relationship which will now be explained.

The expanded sum of products (1) contained seven of the sixteen possible combinations of four variables. A sum of the *other nine* combinations

$$(3) \quad \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D \\ + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + ABC\bar{D} + ABCD$$

represents the *complementary* expanded sum of products.

By definition, two expressions are complementary if, whenever one expression equals 1, the other equals 0 and vice versa, both expressions never both equalling 1 or both equalling 0. Of the sixteen possible combinations of four variables, one and only one combination will equal 1 at a given time, the other fifteen combinations equalling 0. Since all of the combinations are included in the two expanded sum of products (1) and (3), and no combination is included in both, one of the sums must equal 1 and the other must equal 0 at all times. Thus, the two expanded sums of products are complementary.

For example, if $\bar{A} = 1$, $\bar{B} = 1$, $C = 1$, and $\bar{D} = 1$, the first term in the original expanded sum of products equals 1, and the other six terms equal 0. Also, the nine terms in the complementary expanded sum of products equal 0. Thus, in this case the original sum equals 1 and the complementary sum equals 0.

If the complementary expanded sum of products (3) is complemented using DeMorgan's theorem, the expanded product of sums (2), equivalent to the original expanded sum of products (1), is obtained. Each sum in the expanded product of sums (2) is the complement of a combination (product) missing from the original expanded sum of products (1).

The expanded product of sums can therefore be obtained from the expanded sum of products by the complementation of the sum of all the missing products. Also, the expanded sum of products can be obtained from the expanded product of sums by the complementation of the product of all the missing sums.

Note that the *number of products* in the expanded sum of products *plus* the *number of sums* in the expanded product of sums *equals* 2^n , the total number of combinations of n variables. In the preceding example, for

instance, there were seven products in the expanded sum of products and nine sums in the expanded product of sums; seven plus nine equals sixteen, the total number of combinations of four variables.

The expanded sum of products is also referred to as the "standard sum," "canonical sum," "disjunctive normal form," and "minterm canonical form"; each product in the expanded sum of products is also referred to as a "minterm." Similarly, the expanded product of sums is referred to as the "standard product," "canonical product," "conjunctive normal form," and "maxterm canonical form"; each sum in the expanded product of sums is referred to as a "maxterm."

" Minimum " Boolean Expressions

A "minimum" sum of products form of a Boolean expression may be defined in several different ways. Some definitions follow.

- (1) A minimum sum of products is one that contains the minimum number of literals.
- (2) A minimum sum of products is one that contains the minimum number of terms (products).
- (3) A minimum sum of products is one in which the number of literals plus the number of terms minus the number of one-literal terms is a minimum.

A "minimum" product of sums can be analogously defined in terms of literals and factors (sums).

As will be seen later, in logical circuit design, the type of minimum expression desired is dependent upon the type of logical circuitry being used. For instance, if relay circuits are being used, the criterion may be the minimum number of literals. For transistor circuits, the minimum number of terms or factors may be desired. For diode circuits, the third definition would give the minimum number of logic block inputs.¹

Often, but not always, the expression that satisfies one definition will satisfy all definitions. For instance, in one case an expression containing the minimum number of literals will also contain the minimum number of terms; in another case, an expression with the minimum number of literals will contain more terms than another equivalent expression with more literals.

Frequently, in the design of logical circuits a choice is made between a minimum sum of products and a minimum product of sums no matter

¹Except for the special case of a single product, in which case the one term is not counted.

what the criterion is for the minimum. For example, if an expression with a minimum number of literals is desired, a sum of products with the minimum number of literals and a product of sums with the minimum number of literals are compared, and the expression with the smaller number of literals is chosen for circuit implementation.

As an example, the minimum sum of products

$$\bar{A}\bar{C}\bar{D} + A\bar{B}D + A\bar{C}$$

would probably be chosen for circuit implementation in preference to the equivalent minimum product of sums

$$(\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{C} + D)(A + C)(A + \bar{D})$$

A minimum sum of products can be obtained from a sum of products by the application of the simplification theorems. A minimum product of sums can be similarly obtained from a product of sums. In practice, if the expression is quite complex it may not be easy to obtain a minimum by algebraic manipulation, or, even if a minimum is obtained, one may not be sure that it is a minimum. However, other methods of simplification, which will be taken up in Chapters 6 and 7, lead more systematically to a minimum.

A sum of products can be obtained from an expression not already in this form by simply "multiplying out" the expression, that is, applying Theorem 9a in reverse.

A product of sums can be similarly obtained by the dual operation of "adding out," that is, applying Theorem 9b in reverse. However, since "multiplying out" is a more familiar operation than "adding out" (it being permissible in ordinary algebra), the following method for obtaining a product of sums form may be preferred.

- (1) Obtain a sum of products form by multiplying out.
- (2) Complement this expression, using DeMorgan's theorem. This complement will be in a product of sums form.
- (3) Multiply out this complement. The complement will now be in a sum of products form.
- (4) Complement this complemented sum of products form, using DeMorgan's theorem. Since the complement of a complement is equivalent to the original, a product of sums form equivalent to the original expression is obtained.

The procedure may be modified by utilizing the *dual* rather than the complement. Starting with a sum of products form, obtain the dual expression. The dual will be in a product of sums form. Multiply out the dual to get it in a sum of products form. Finally, obtain the dual of the dual to get a product of sums equivalent to the original. By using the dual, rather

than the complement, it is not necessary to complement all literals twice in the procedure.

For an example, a product of sums will be obtained from the sum of products,

$$\bar{A}C\bar{D} + A\bar{B}D + A\bar{C}$$

Obtain the dual:

$$(\bar{A} + C + \bar{D})(A + \bar{B} + D)(A + \bar{C})$$

Multiply out the dual (Some obvious simplifications can be made in the process.):

$$\bar{A}\bar{B}\bar{C} + \bar{A}\bar{C}D + AC + A\bar{D} + \bar{B}\bar{C}\bar{D}$$

Obtain the dual of the dual to get the product of sums:

$$(\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{C} + D)(A + C)(A + \bar{D})(\bar{B} + \bar{C} + \bar{D})$$

If a minimum product of sums is desired, the above expression is examined for further simplification, and it is found that either the first or last factor is redundant. It is suggested that the reader verify this for practice. There are thus two minimum product of sums:

$$(\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{C} + D)(A + C)(A + \bar{D})$$

$$(\bar{A} + \bar{C} + D)(A + C)(A + \bar{D})(\bar{B} + \bar{C} + \bar{D})$$

Application of the simplification theorems can, of course, be made while the expression is still in the dual sum of products form.

By the judicious selection of pairs of factors to multiply together, the amount of work involved in the multiplying out process can be considerably reduced. In general, it is desirable to multiply together factors with variables in common, complemented or not.

For example, in the expression

$$(A + B + C)(D + E)(A + B + F)(\bar{D} + G)$$

multiplying together the first two factors, and multiplying together the last two factors, gives as a first step

$$(AD + AE + BD + BE + CD + CE)(A\bar{D} + AG + B\bar{D} + BG + \bar{D}F + FG)$$

whereas, multiplying together the first and third factors (variables A and B common), and multiplying together the second and fourth factors (variable D common), gives as a first step

$$(A + B + CF)(DG + \bar{D}E)$$

Earlier in the chapter, an expanded product of sums (2), was obtained from the first of the two minimum product of sums above. It is suggested that for practice the reader expand the second minimum product of sums, and verify that the same expanded product of sums is obtained.

Minimum Factored Form

A minimum sum of products or a minimum product of sums can generally be factored. The minimum factored form is that expression with the absolute minimum number of literals. This form is often desirable for relay circuit implementation.

There is no formal method for always obtaining the minimum factored form. A minimum sum of products or minimum product of sums can be factored in all possible ways, and the solution with the minimum number of literals selected. However, it may be possible to add redundancy before factoring, and obtain an expression with fewer literals.

EXAMPLE:

If the expression

$$VW + VX + W\bar{Y} + WZ + XYZ$$

is factored in all possible ways, the expression with the minimum number of literals is found to be

$$X(V + YZ) + W(V + \bar{Y} + Z) \quad (8 \text{ literals})$$

However, if a redundant Y is added to the WZ term, giving

$$VW + VX + W\bar{Y} + WYZ + XYZ$$

the minimum factored form

$$(X + W)(V + YZ) + W\bar{Y} \quad (7 \text{ literals})$$

is obtained.

Functions of n Variables

With n variables there are 2^n possible combinations, and these combinations can form $2^{(2^n)}$ different functions. For example, with two variables X and Y , there are four possible combinations:

$$\begin{array}{c} \bar{X}\bar{Y} \\ \bar{X}Y \\ X\bar{Y} \\ XY \end{array}$$

and these combinations can form sixteen different functions. These functions are shown, arranged in two columns so that each row contains a complementary pair.

Sixteen Functions of Two Variables

	0	$\bar{X}\bar{Y} + \bar{X}Y + X\bar{Y} + XY = 1$
	$XY = XY$	$\bar{X}\bar{Y} + \bar{X}Y + X\bar{Y} = \bar{X} + \bar{Y}$
	$X\bar{Y} = X\bar{Y}$	$\bar{X}\bar{Y} + \bar{X}Y + XY = \bar{X} + Y$
$\bar{X}Y$	$= \bar{X}Y$	$\bar{X}\bar{Y} + X\bar{Y} + XY = X + \bar{Y}$
$\bar{X}\bar{Y}$	$= \bar{X}\bar{Y}$	$\bar{X}Y + X\bar{Y} + XY = X + Y$
	$X\bar{Y} + XY = X$	$\bar{X}\bar{Y} + \bar{X}Y = \bar{X}$
	$\bar{X}Y + XY = Y$	$\bar{X}\bar{Y} + X\bar{Y} = \bar{Y}$
$\bar{X}\bar{Y}$	$+ XY = \bar{X}\bar{Y} + XY$	$\bar{X}Y + X\bar{Y} = \bar{X}Y + X\bar{Y}$

The table below gives a few corresponding values of n , 2^n , and $2^{(2^n)}$. Note that if the number of variables is increased by one, the number of functions is squared.

<i>Number of Variables</i>	<i>Number of Combinations</i>	<i>Number of Functions</i>
n	2^n	$2^{(2^n)}$
0	1	2
1	2	4
2	4	16
3	8	256
4	16	65,536
5	32	4,294,967,296

PROBLEMS

- 1. Minimum sum of products: $\bar{A} + B\bar{C}$. Express as:
 - (a) minimum product of sums (making use of dual).
 - (b) expanded sum of products.
 - (c) expanded product of sums.
- 2. Express $A(B + \bar{C}) + D$ as:
 - (a) minimum sum of products.
 - (b) minimum product of sums.
 - (c) expanded sum of products.
 - (d) expanded product of sums.
- *3. Minimum sum of products: $A\bar{C} + \bar{A}D + B\bar{D}$. Express as:
 - (a) minimum product of sums.

- (b) expanded sum of products.
- (c) expanded product of sums.

***4.** Express $\bar{B}C + \bar{B}D + \bar{A}C + A\bar{D}$ as:

- (a) minimum sum of products.
- (b) minimum product of sums.
- (c) expanded sum of products.
- (d) expanded product of sums.

3

Logical Circuits

So far, Boolean variables have been related to basic statements that could be either true or false. If a basic statement were true, its corresponding variable was equal to 1; if a statement were false, the corresponding variable was equal to 0. An entire Boolean expression was related to a compound statement that was either true or false. If the Boolean expression equalled 1, the compound statement was true; if the expression equalled 0, the compound statement was false.

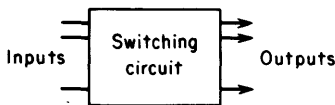


Figure 3-1

Now, the relationship of Boolean algebra to switching circuits will be discussed. A switching circuit may be thought of in terms of the schematic diagram in Fig. 3-1. Each input and output line must be in one of two possible states at any given time. For instance, a line may be "on" or "off," "high" or "low," "+" or "-", "conducting" or "not conducting."

For the present time, the discussion will be limited to *combinational*

circuits, in which the state of an output is determined solely by the states of the inputs. Switching circuits having only a single output will be discussed at this time.

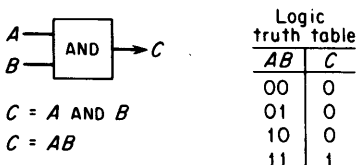
The output of a logical circuit is a function of its inputs. A Boolean expression is a function of its variables. The output conditions of a logical circuit can thus be represented by a Boolean expression, each input being represented by a variable. Simplification of a Boolean expression is thus related to the simplification of the corresponding logical circuit.

Switching circuits are made up of interconnections of basic "logic blocks." Five common functions that are performed by logic blocks used in computers and other similar equipment are the AND, OR, NOT, NAND, and NOR functions. These functions will now be presented, and a diagram symbol and logic truth table shown for each.

AND Function

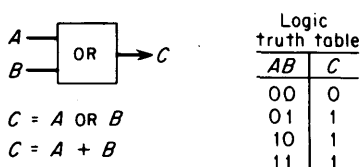
The function (Fig. 3-2) equals 1 only when *all* of the variables equal 1. Conversely, the function equals 0 only when *one or more* of the variables equal 0.

For simplicity, a two-variable AND function is used for illustration, but it should be recognized that the function also applies to more than two variables. The same applies to the OR, NAND, and NOR functions.



AND Function

Figure 3-2



OR Function

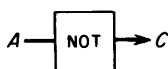
Figure 3-3

OR Function

The function (Fig. 3-3) equals 1 only when *one or more* of the variables equal 1. Conversely, the function equals 0 only when *all* of the variables equal 0.

NOT Function

The function (Fig. 3-4) equals 1 only when the (single) variable equals 0. Conversely, the function equals 0 only when the (single) variable equals 1.



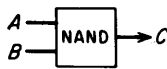
$$C = \text{NOT } A$$

$$C = \bar{A}$$

Logic truth table	
A	C
0	1
1	0

NOT Function

Figure 3-4



$$C = \text{NOT } (A \text{ AND } B)$$

$$C = \overline{AB} = \bar{A} + \bar{B}$$

Logic truth table	
AB	C
00	1
01	1
10	1
11	0

NAND Function

Figure 3-5

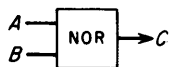
NAND Function

This is a COMPLEMENTED AND or NOT AND function: NOT (A AND B). "NAND" is a contraction of "NOT AND." The function (Fig. 3-5) equals 0 only when *all* of the variables equal 1. Conversely, the function equals 1 only when *one or more* of the variables equal 0.

The NAND function is also called the *Sheffer Stroke* function, and its Boolean symbol is $|$. For example, $A|B$ is equivalent to \overline{AB} .

NOR Function

This is a COMPLEMENTED OR or NOT OR function: NOT (A OR B). "NOR" is a contraction of "NOT OR."



$$C = \text{NOT } (A \text{ OR } B)$$

$$C = \overline{A+B} = \bar{A}\bar{B}$$

Logic truth table	
AB	C
00	1
01	0
10	0
11	0

NOR Function

Figure 3-6

The function (Fig. 3-6) equals 0 only when *one or more* of the variables equal 1. Conversely, the function equals 1 only when *all* of the variables equal 0.

The NOR function is also called the *Pierce Arrow* function, and its Boolean symbol is \downarrow . For example, $A\downarrow B$ is equivalent to $\overline{A+B}$.

Logic Blocks

Logic blocks that perform the functions just described will now be discussed, the discussion being limited, at this time, to how the blocks behave logically. What is in the blocks and why they behave as they do will be considered in the next chapter.

For purposes of reference, consider the input and output lines of the logic blocks as being at one of two possible voltage levels. The actual value of the two levels is not important here; "+" will be used to represent the more positive of the two levels, and "-" to represent the more negative.

An input may be related to its corresponding variable in either of two ways:

input: + = corresponding variable: 1
input: - = corresponding variable: 0

or

input: - = corresponding variable: 1
input: + = corresponding variable: 0

An output can be similarly related to its corresponding Boolean expression in either of two ways:

output: + = corresponding expression: 1
output: - = corresponding expression: 0

or

output: - = corresponding expression: 1
output: + = corresponding expression: 0

A circuit output requirement is usually specified in terms of the conditions for which the corresponding Boolean expression equals 1, and this convention will be used throughout.

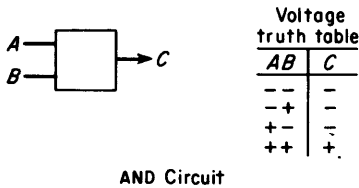
For the time being, consider only the

$$\begin{aligned} + &= 1 \\ - &= 0 \end{aligned}$$

assignment at both the output and inputs. With this assignment, a circuit is said to use “positive logic.” It is customary to name logic blocks according to the function they perform with positive logic; hence, the nomenclature of the blocks to be discussed.

AND Circuit

The first logic block has the following characteristic: the output is + only when *all* of the inputs are +; conversely, the output is - only when *one or more* of the inputs are -. The “voltage truth table” for this logic block is shown in Fig. 3-7.



AND Circuit
Figure 3-7

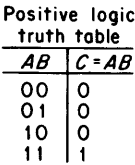
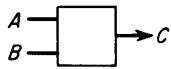


Figure 3-8

The positive logic truth table for this block is shown in Fig. 3-8. It can be seen from the truth table (Fig. 3-8) that, with positive logic, this block performs the AND function; such blocks are therefore called AND circuits.

OR Circuit

The next logic block has the following characteristic: the output is + only when *one or more* of the inputs are +; conversely, the output is - only when *all* of the inputs are -. The voltage truth table and positive logic truth table for this logic block are shown in Fig. 3-9. It can be seen from

	Voltage truth table		Positive logic truth table	
	AB	C	AB	$C=A+B$
	--	-	00	0
	-+	+	01	1
	+-	+	10	1
	++	+	11	1

OR Circuit

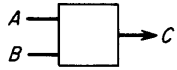
Figure 3-9

the positive logic truth table that, with positive logic, this block performs the OR function; therefore, these blocks are called OR circuits.

The naming of the next two logic blocks follows the same reasoning.

NAND Circuit

The output is - only when *all* of the inputs are +; conversely, the output is + only when *one or more* of the inputs are - (Fig. 3-10).

	Voltage truth table		Positive logic truth table	
	AB	C	AB	$C=\overline{AB}=\overline{A}+\overline{B}$
	--	+	00	1
	-+	+	01	1
	+-	+	10	1
	++	-	11	0

NAND Circuit

Figure 3-10

NOR Circuit

The output is - only when *one or more* of the inputs are +; conversely, the output is + only when *all* of the inputs are - (Fig. 3-11).

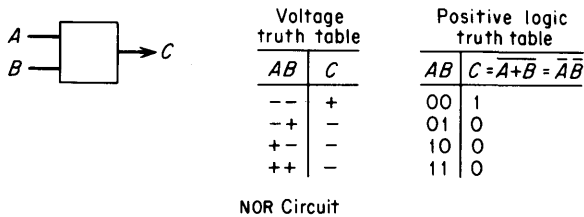


Figure 3-11

NOT Circuit—Inverter

This logic block (Fig. 3-12) has only a single input. The output is + only when the input is - ; conversely, the output is - only when the input is + . Although this block performs logical complementation, that is, the NOT function, it is commonly referred to as an *inverter*.

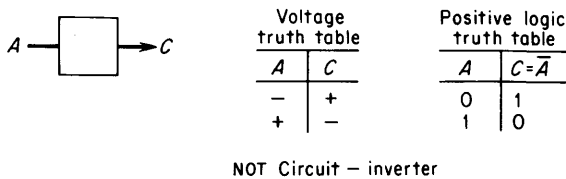


Figure 3-12

Negative Logic

If the opposite assignment,

- = 1

+ = 0

is made, a circuit is said to use “negative logic.” Referring to the logic blocks previously described, with negative logic:

- an AND circuit performs the OR function
- an OR circuit performs the AND function
- a NAND circuit performs the NOR function
- and a NOR circuit performs the NAND function

A NOT circuit performs the NOT function regardless of whether positive or negative logic is used. The functions of these logic blocks will now be examined in more detail.

In the diagramming of logical circuits, the practice followed here is that

the name of the *function* performed, rather than the name of the logic block itself, is written in the diagram symbol. Negative logic assigned to an input or output line is identified by a small circle drawn at the junction of the line and the symbol.

AND Circuit

Reference to the negative logic truth table (Fig. 3-13) shows that, with negative logic, an AND circuit performs the OR function. Note that the voltage truth table for a particular logic block remains fixed; it is the assignment of the “+” and “−” that prescribes the function that the logic block performs.

Voltage truth table	Positive logic truth table	Negative logic truth table
$\begin{array}{c c c} A & B & C \\ \hline \end{array}$	$\begin{array}{c c c} A & B & C=AB \\ \hline \end{array}$	$\begin{array}{c c c} A & B & C=A+B \\ \hline \end{array}$
$\begin{array}{c c c} - & - & - \\ \hline \end{array}$	$\begin{array}{c c c} 0 & 0 & 0 \\ \hline \end{array}$	$\begin{array}{c c c} 1 & 1 & 1 \\ \hline \end{array}$
$\begin{array}{c c c} - & + & - \\ \hline \end{array}$	$\begin{array}{c c c} 0 & 1 & 0 \\ \hline \end{array}$	$\begin{array}{c c c} 1 & 0 & 1 \\ \hline \end{array}$
$\begin{array}{c c c} + & - & - \\ \hline \end{array}$	$\begin{array}{c c c} 1 & 0 & 0 \\ \hline \end{array}$	$\begin{array}{c c c} 0 & 1 & 1 \\ \hline \end{array}$
$\begin{array}{c c c} + & + & + \\ \hline \end{array}$	$\begin{array}{c c c} 1 & 1 & 1 \\ \hline \end{array}$	$\begin{array}{c c c} 0 & 0 & 0 \\ \hline \end{array}$

AND Function	OR Function
$\begin{array}{c} A \\ B \end{array} \rightarrow \boxed{\text{AND}} \rightarrow C$	$\begin{array}{c} A \\ B \end{array} \rightarrow \boxed{\text{OR}} \rightarrow C$
AND Circuit	

Figure 3-13

EXAMPLE:

- (a) An AND circuit is used to perform the AND function AB . Positive logic must be used. The expression AB equals 1 if A equals 1 AND B equals 1. The output is + if A is + AND B is + (Fig. 3-14).

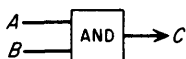


Figure 3-14

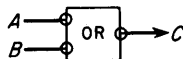


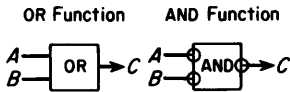
Figure 3-15

- (b) An AND circuit is used to perform the OR function $A + B$. Negative logic must be used. The expression $A + B$ equals 1 if A equals 1 OR B equals 1. The output is − if A is − OR B is − (Fig. 3-15).

Note that both diagrams above denote the same logic block—an AND circuit. Following are summaries of the OR, NAND, NOR, and NOT circuits, and the functions they perform with positive and negative logic.

OR Circuit

Voltage truth table		Positive logic truth table		Negative logic truth table	
AB	C	AB	$C=A+B$	AB	$C=AB$
--	-	00	0	11	1
-+	+	01	1	10	0
+-	+	10	1	01	0
++	+	11	1	00	0

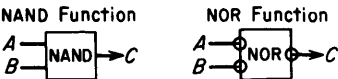


OR Circuit

Figure 3-16

NAND Circuit

Voltage truth table		Positive logic truth table		Negative logic truth table	
AB	C	AB	$C=\overline{AB}=\overline{A+B}$	AB	$C=\overline{A+B}=\overline{AB}$
--	+	00	1	11	0
-+	+	01	1	10	0
+-	+	10	1	01	0
++	-	11	0	00	1

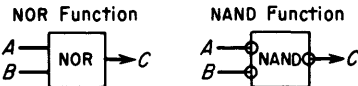


NAND Circuit

Figure 3-17

NOR Circuit

Voltage truth table		Positive logic truth table		Negative logic truth table	
AB	C	AB	$C=\overline{A+B}=\overline{AB}$	AB	$C=\overline{AB}=\overline{A+B}$
--	+	00	1	11	0
-+	-	01	0	10	1
+-	-	10	0	01	1
++	-	11	0	00	1



NOR Circuit

Figure 3-18

NOT Circuit—Inverter

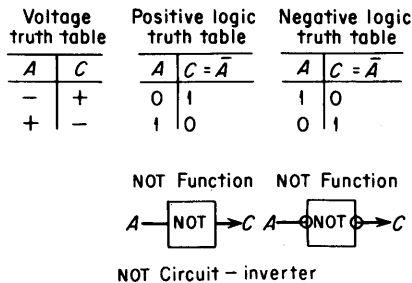


Figure 3-19

An inverter functions the same regardless of the type of logic used.

Application of Positive and Negative Logic

To further illustrate the application of positive and negative logic, the design of a simple circuit will be examined using both types of logic.

Circuit requirement:

Three inputs: A (Storage loaded)
 B (Error)
 C (Computation finished)

Output: If storage loaded, or if no error and computation finished.
 Boolean expression for output:

$$A + \bar{B}C$$

Positive logic implementation (Fig. 3-20):

If storage loaded, input A is +.

If error, input B is +.

If computation finished, input C is +.

If storage loaded, or no error and computation finished, output is +.

Output is + if A is +, or if B is - and C is +.

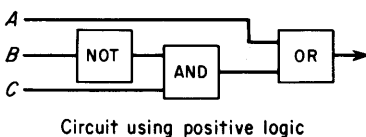


Figure 3-20

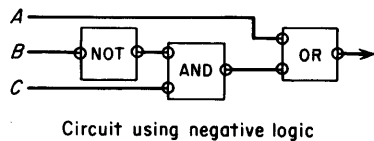


Figure 3-21

Negative logic implementation (Fig. 3-21):

- If storage loaded, input *A* is −.
- If error, input *B* is −.
- If computation finished, input *C* is −.
- If storage loaded, or no error and computation finished, output is −.
- Output is − if *A* is −, or if *B* is + and *C* is −.

Since negative logic is being used, an OR circuit is used to perform the AND function, and an AND circuit is used to perform the OR function.

Mixed Logic

Sometimes “mixed logic” is used, that is, positive logic is used for the inputs, and negative logic for the output, or vice versa. For instance, if with a NAND circuit, positive logic is used at the inputs and negative logic is used at the output, the AND function is performed (Fig. 3-22).

With the option of employing positive or negative logic at the inputs, and positive or negative logic at the output, the AND, OR, NAND, and NOR circuits each can perform the AND, OR, NAND, or NOR functions. Figure 3-23 is a summary of the functions performed by these logic blocks with all combinations of positive and negative logic.

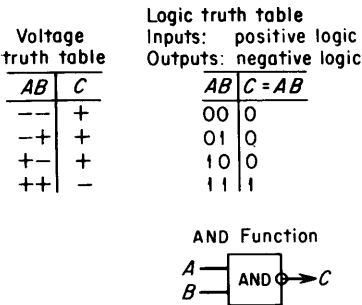


Figure 3-22

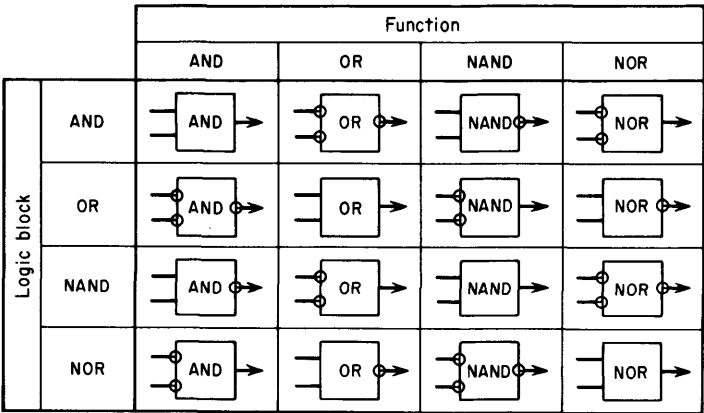


Figure 3-23

Suppose it is desired to implement the AND function. Using an AND circuit the output is + only when *all* inputs are +. Using an OR circuit the output is - only when *all* inputs are -. Using a NAND circuit the output is - only when *all* inputs are +. Using a NOR circuit the output is + only when *all* inputs are -. The other functions can be similarly analyzed.

For illustration of some of the variations possible in logic implementation, the "exclusive or" function $A\bar{B} + \bar{A}B$ will now be examined. Positive logic will be used throughout.

Using AND, OR, and NOT functions, the "exclusive or" can be realized as in Fig. 3-24.

A more economical realization can be obtained (Fig. 3-25) by recognizing the following Boolean identities.

$$\begin{aligned} A\bar{B} + \bar{A}B &= (A + B)(\bar{A} + \bar{B}) \\ &= (A + B)\overline{AB} \end{aligned}$$

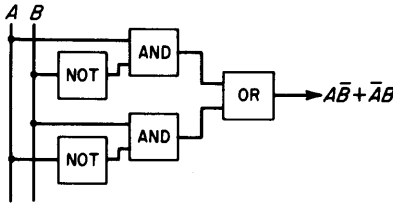


Figure 3-24

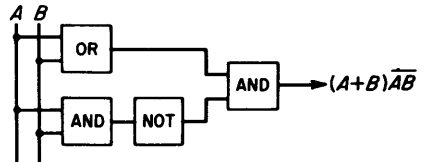


Figure 3-25

Any Boolean function can be realized with NAND functions only or with NOR functions only.¹ The NOT function is obtained by using only one input of the logic block. Figure 3-26 shows the realization of the "exclusive or" function using NAND functions only. Figure 3-27 is a more economical realization.

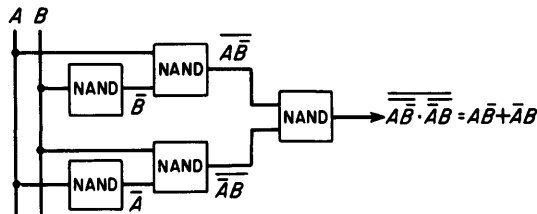


Figure 3-26

¹Any Boolean function can also be realized with AND and NOT functions only, or with OR and NOT functions only.

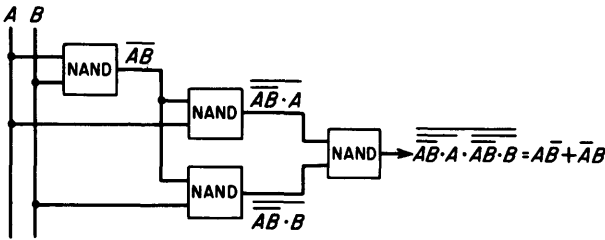


Figure 3-27

It is interesting to note that a two-stage NAND circuit is equivalent to an AND-OR circuit (Fig. 3-28). Also, a two-stage NOR circuit is equivalent to an OR-AND circuit (Fig. 3-29).

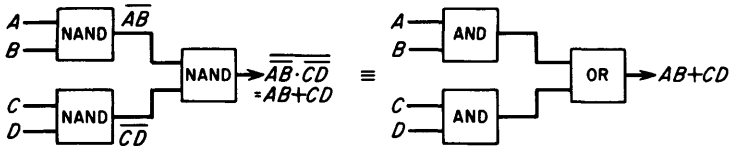


Figure 3-28

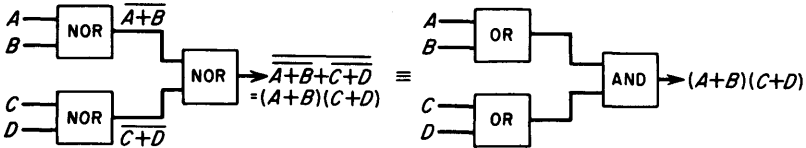


Figure 3-29

The Boolean expression corresponding to the most economical circuit implementation generally varies with the types of logic blocks used. Also, the various types of logic blocks themselves generally differ in cost. In multi-output circuits, where the outputs can “share” logic blocks in common, the most economical over-all circuit implementation may not be made up of the most economical implementations for each individual output. Furthermore, a given logic block has physical limits on the number of allowable inputs, and on the types and number of other logic blocks that can be “driven” from its output. Therefore, experience and ingenuity are often helpful in arriving at the most economical logical circuit.

4

Electronic Logic Blocks

In the previous chapter, some common logic blocks were studied from the functional standpoint. The internal structure of some of these blocks will now be examined to see why they function as they do. Since logic rather than “hardware,” is of primary interest, and also because of the ever-changing technology, no attempt is made here to examine any more than a representative sample of existing logic blocks.

Most blocks will be illustrated with only two inputs. However, it should be understood that a greater number of inputs is generally possible.

Diode Logic Blocks

Figures 4-1 and 4-2 show two logic blocks employing diodes.

In the first circuit, the output will be at the same voltage level as the lowest input voltage level. Therefore, the output voltage level is + only if *all* input voltage levels are +, and this is an AND circuit.

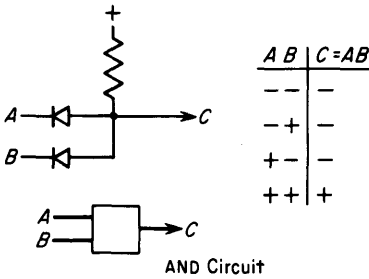


Figure 4-1

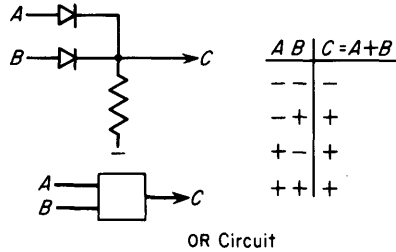


Figure 4-2

In the second circuit, the output will be at the same voltage level as the highest input voltage level. Therefore, the output is + if *any* input is +, and this is an OR circuit.

Vacuum Tube Logic Blocks

Figures 4-3 through 4-6 show logic blocks employing vacuum tubes.

In the first circuit (Fig. 4-3), which uses a single vacuum tube triode, if the grid input is at the low voltage level, conduction through the triode is prevented, no current flows through the load resistor, and the output is at the high voltage level. If the grid input is at the high voltage level, conduction takes place through the triode, current flows through the load resistor causing a voltage drop across it, and the output is at the low voltage level. Thus, this circuit is an inverter.

In the second circuit (Fig. 4-4), multiple triodes have their plates connected to a common load resistor. If *all* grid inputs are at the low voltage level none of the triodes conduct, there is no current through the load

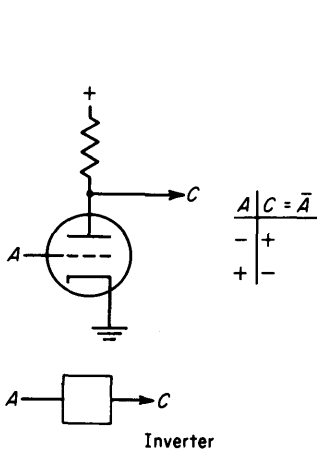


Figure 4-3

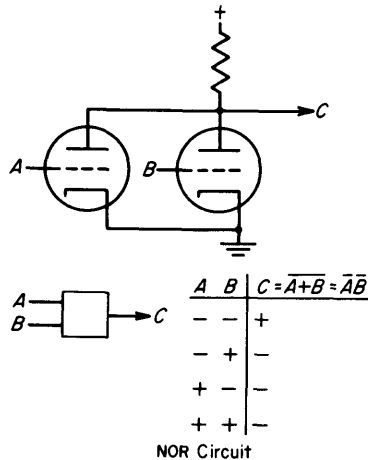
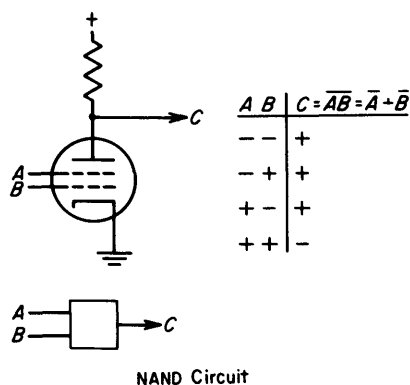


Figure 4-4

resistor, and the output is at the high level. If *any* grid input is high there will be conduction through that triode, current will flow through the load resistor causing a voltage drop across it, and the output will be at the low voltage level. Thus, this is a NOR circuit.

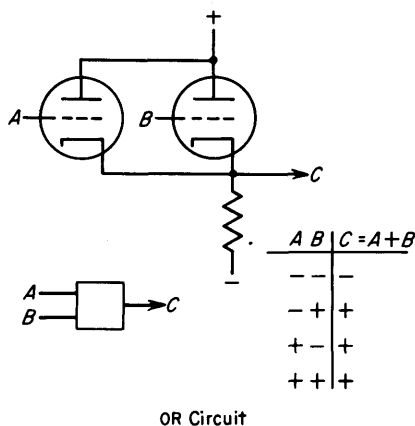
In the third circuit (Fig. 4-5), a pentode is used. (This vacuum tube has a third grid which is not shown.) If either grid input is at the low voltage level, conduction through the pentode is prevented and the output is at the high voltage level. If both grid inputs are at the high voltage level, the pentode conducts and the output is at the low voltage level. Thus, this is a NAND circuit.

In the fourth circuit (Fig. 4-6), multiple triodes have their cathodes



NAND Circuit

Figure 4-5



OR Circuit

Figure 4-6

connected to a common load resistor. If *all* grid inputs are at the low voltage level none of the triodes conduct, no current flows through the load resistor, and the output is at the low voltage level. If *any* grid input is at the high level there will be conduction through that triode, current will flow through the load resistor causing a voltage drop across it, and the output will be at the high voltage level. Thus, this is an OR circuit.

A few more vacuum tube circuits are shown to illustrate some of the flexibility possible with vacuum tube logical circuits (Fig. 4-7). It may be helpful in analyzing such circuits to first write the logical expression describing the conditions for which current flows through the load resistor, and then determine whether the output is + or - for this conduction. If it is +, the output expression is equivalent to that for conduction; if it is -, the output expression is the complement of that for conduction. For example, in the first of the two circuits above, there is conduction through the load resistor if the triode or the pentode conducts. The triode conducts if A is +; the pentode conducts if B and C are +. Therefore, the expression for con-

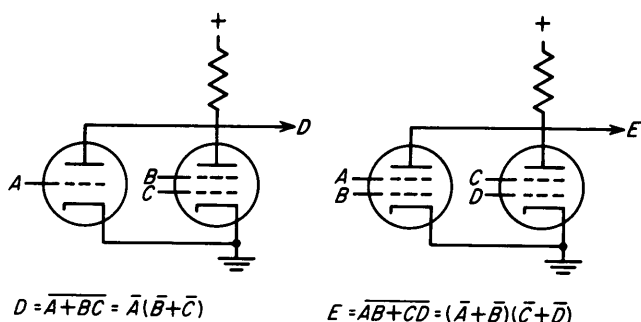


Figure 4-7

duction in the load resistor is $A + BC$. The complement of this expression, $\overline{A + BC}$, describes the conditions for a + output. The second circuit can be analyzed in a similar manner.

Transistor Logic Blocks

The versatility of transistors will be seen in the study of the logic blocks that follow.

There are two basic types of transistors: nPn and pNp . The nPn transistor (Fig. 4-8) is analogous to a vacuum tube triode. The collector is analogous to the plate. The emitter is analogous to the cathode. The base is analogous to the grid. A + base allows conduction, electron flow being from emitter to collector. The pNp transistor (Fig. 4-9) is analogous to a hypothetical

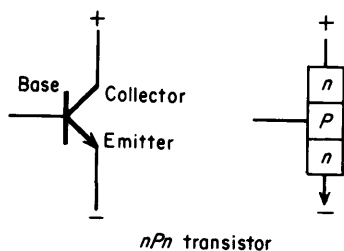


Figure 4-8

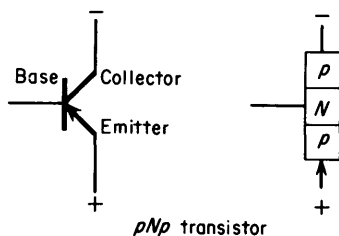


Figure 4-9

vacuum tube triode that operates with all voltages reversed. A — base allows conduction, electron flow being from collector to emitter.

An inverter is made by placing a load resistor on the collector side of the transistor; an emitter-follower is made by placing the resistor on the emitter side. These circuits are shown in Figs. 4-10 and 4-11. Note that single emitter-followers do not perform a logical function. However, when they

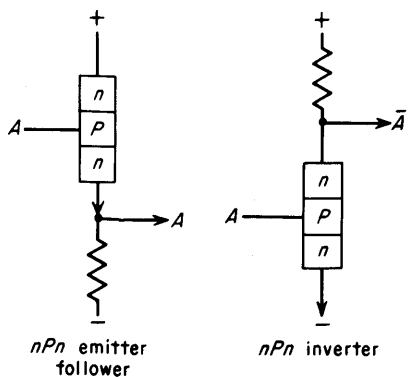


Figure 4-10

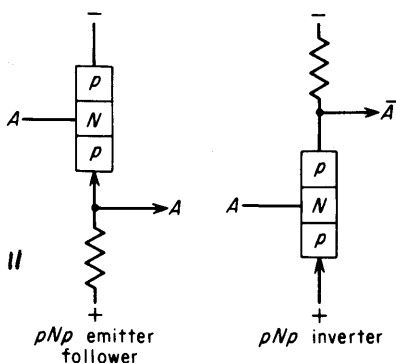


Figure 4-11

are used in multiples, or with other circuits, they do perform logical functions. Some logical circuits using emitter-followers and inverters are shown in Fig. 4-12.

These circuits can be analyzed similarly to the vacuum tube circuits previously discussed. For example, in the last circuit of Fig. 4-12, there is conduction through the load resistor if A is $-$ or B is $+$. The output is $+$ when there is no conduction; therefore, the expression for a $+$ output is $\bar{A} + B = A\bar{B}$.

A summary of the types of logical circuits obtainable with these transistor emitter-followers and inverters is shown in the table below.

			Input B			
			Emitter-Follower		Inverter	
			nPn	pNp	nPn	pNp
Input A	Emitter-Follower	nPn	$A + B$			$A + \bar{B}$
		pNp		AB	$A\bar{B}$	
	Inverter	nPn		$\bar{A}B$	$\bar{A}\bar{B}$	
		pNp	$\bar{A} + B$			$\bar{A} + \bar{B}$

Another type of transistor logic block has complementary outputs (Fig. 4-13). "P-blocks," made up of nPn transistors, have OR and NOR outputs, and "N-blocks," made up of pNp transistors, have AND and NAND outputs.

Still another type of transistor logic block uses resistors or diodes in conjunction with a transistor, the resistors or diodes performing an AND

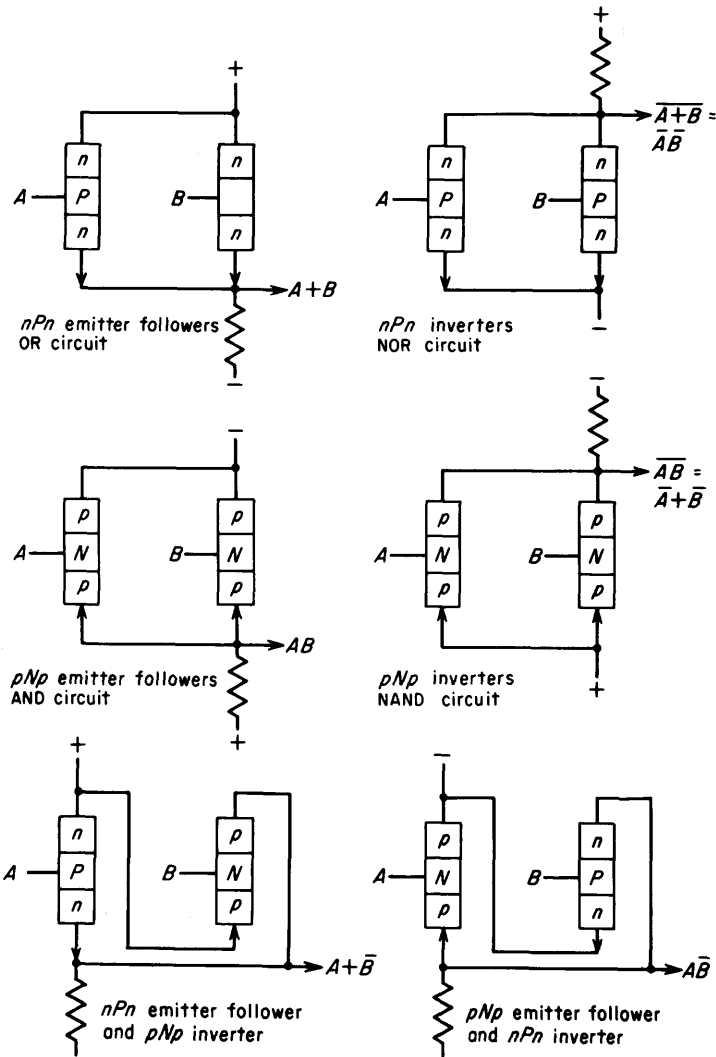


Figure 4-12

or OR function, with a transistor inverter complementing the output. These logic blocks are thus NAND or NOR circuits. Figure 4-14 shows some logic blocks of this type.

The logic capabilities of transistor logic blocks are often extended by commoning collectors; this is sometimes referred to as "dotting." "Dotting" any of the logic blocks shown in Fig. 4-14 results in a second stage of logic,

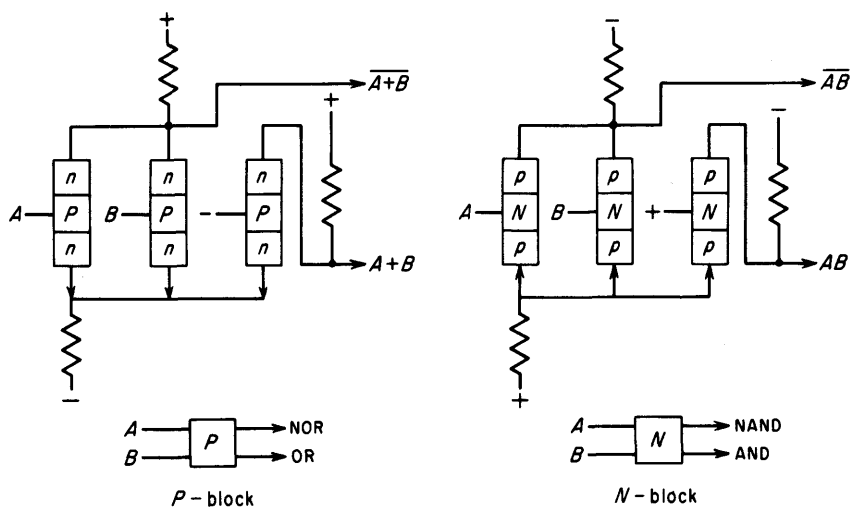
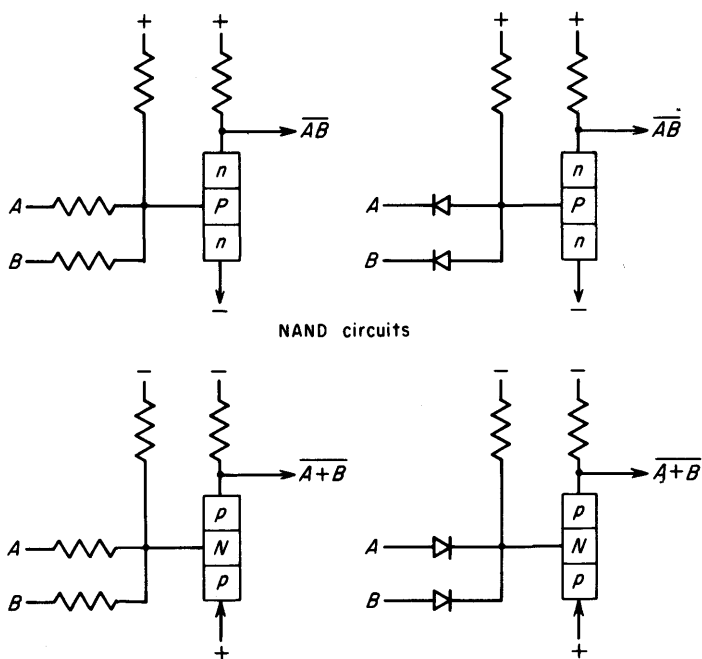


Figure 4-13



NOR circuits

Figure 4-14

as illustrated in Fig. 4-15. Some other transistor logic blocks are shown in Fig. 4-16. "Dotting" these logic blocks serves only to expand the input limit, as illustrated in Fig. 4-17.

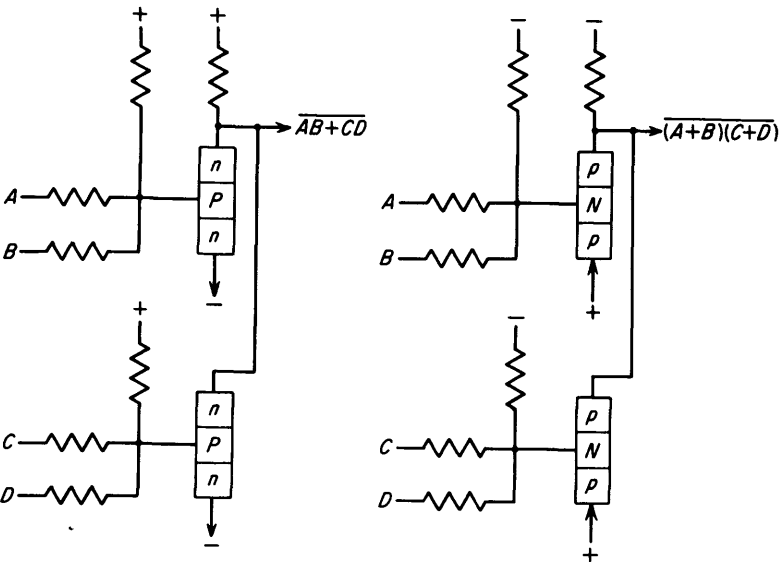


Figure 4-15

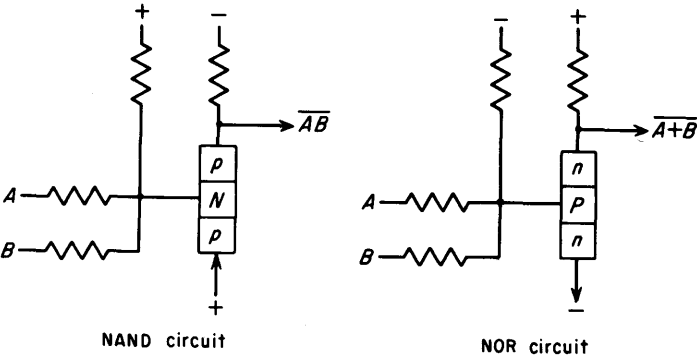


Figure 4-16

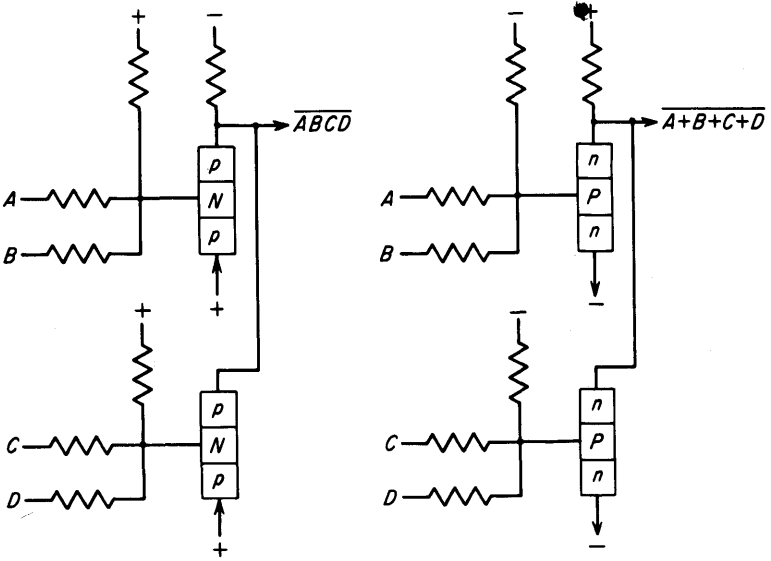


Figure 4-17

5

Contact Networks

In this chapter, the relationship of Boolean expressions and contact networks will be examined. Contacts may be operated by several means such as switches, keys, cams, or relays. The following discussion will be exclusively in terms of relays, which can operate a number of contacts simultaneously. The implementation of simpler devices, such as switch contacts, will be obvious.

The general schematic diagram for a switching circuit (Fig. 5-1) can still hold for relay contact networks if the circuit is thought of as it appears in Fig. 5-2. For the time being, only contact networks with a single output will be considered.

In electronic circuits, the inputs and outputs are thought of as being at one of two possible voltage levels. In contact networks, the individual contacts and the entire contact networks are thought of as being either *closed* or *open*; again, there are exactly two possible states. In relating a Boolean expression, which may equal 1 or 0, to a contact network, which may be *closed* or *open*, the following assignment is usually made.

Boolean Expression		Contact Network
1	=	closed
0	=	open

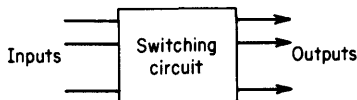


Figure 5-1

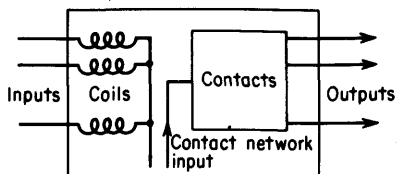


Figure 5-2

In a Boolean expression, the variables, which may equal 1 or 0, relate to the relays, which may be *operated* or *unoperated*, as follows.

Boolean Variable		Relay
1	=	operated
0	=	unoperated

For example, the Boolean expression $A + \bar{B}C$ relates to a relay contact network as follows: $A + \bar{B}C = 1$ if $A = 1$ or if $B = 0$ and $C = 1$. The related contact network is closed if relay A is operated or if relay B is unoperated and relay C is operated.



Figure 5-3



Figure 5-4

The discussion of the implementation of such contact networks will be limited, for the time being, to two types of relay contacts: normally-open, N/O , and normally-closed, N/C . (N/O contacts are also called “make” contacts; N/C contacts are also called “break” contacts.) The unoperated state of a relay is considered the *normal* state. Thus, normally-open contacts are open when the relay is unoperated, and closed when the relay is operated. Normally-closed contacts are closed when the relay is unoperated, and open when the relay is operated. By convention, contact networks are drawn with the contacts shown in their normal state. A N/O contact on relay X might be drawn pictorially as in Fig. 5-3 and a N/C contact on relay X as in Fig. 5-4.

Suppose the following simple circuit requirement: a circuit is to be closed if relay X is operated. The Boolean expression for the circuit requirement is simply

$$X$$

and the circuit would be drawn as in Fig. 5-5. When relay X is operated, the N/O contact is closed.



Figure 5-5



Figure 5-6

Suppose now, another, equally simple circuit requirement: a circuit is to be closed if relay X is unoperated. The Boolean expression for this circuit is

$$\bar{X}$$

and the circuit would be implemented as in Fig. 5-6. When relay X is unoperated, the N/C contact is closed.

In the preceding two examples, note the relationship between an uncomplemented literal and its corresponding N/O contact, and the relationship between a complemented literal and its corresponding N/C contact. This relationship gives the convenient symbolic notation for relay contacts shown in Fig. 5-7. Each uncomplemented literal, in a Boolean expression relating to a contact network, corresponds to a N/O contact; each complemented literal in the expression corresponds to a N/C contact. The relationships thus far established are summarized in the table below.

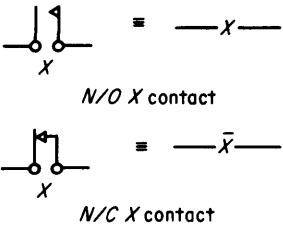


Figure 5-7

Relay	N/O Contact	N/C Contact	Literal	
X	X	\bar{X}	X	\bar{X}
Operated	Closed	Open	1	0
Unoperated	Open	Closed	0	1

Implementation of AND, OR, and NOT Functions

AND

Suppose a relay contact network is to be closed only if relays A AND B are operated. The Boolean expression for this circuit is

$$AB$$

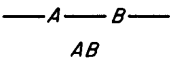


Figure 5-8

and the network requires a N/O contact on relay A and a N/O contact on relay B . For the network to be closed only when both relays A AND B are operated, these contacts must be placed in series (Fig. 5-8). Thus, the Boolean AND function is realized in contact

networks by a series connection.

\cdot = AND = series connection

OR

Suppose that a contact network is to be closed if relay A OR B is operated. The Boolean expression for this circuit is

$$A + B$$

Again, a N/O contact is required on each relay. For the network to be

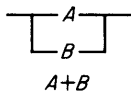


Figure 5-9

closed if relay A OR B is operated, a parallel connection is required (Fig. 5-9). Thus, the Boolean OR function is realized in contact networks by a parallel connection.

$$+ = \text{OR} = \text{parallel connection}$$

NOT

The NOT function, as previously shown, is implemented by the use of normally-closed contacts. Thus, if a relay contact network is to be closed if, say, relay A is NOT operated, the Boolean expression would be

$$\bar{A}$$

and the circuit would be realized by the use of a normally-closed contact on A .

The circuit to realize the function $A + \bar{B}C$, discussed earlier in this chapter, would be as in Fig. 5-10. It should be noted that a Boolean expression is related to a single-input single-output (two-terminal) *series-parallel* contact network.

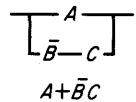


Figure 5-10

The examples in Fig. 5-11 show the application of a few Boolean algebra

$$1. \begin{array}{c} \overline{\overline{A} - B} \\ A + \bar{A}B \end{array} \equiv \begin{array}{c} \overline{A} \\ B \end{array} \quad \text{Theorem 12a}$$

$$A + \bar{A}B = A + B$$

$$2. \begin{array}{c} \overline{A - B} \\ \overline{\bar{A} - C} \\ B - C \end{array} \equiv \begin{array}{c} \overline{A - B} \\ \bar{A} - C \end{array} \quad \text{Theorem 13a}$$

$$AB + \bar{A}C + BC = AB + \bar{A}C$$

$$3. \begin{array}{c} \overline{A - B} \\ \bar{A} - C \end{array} \equiv \begin{array}{c} \overline{A} \\ C \end{array} \overline{\bar{A} - B} \quad \text{Theorem 14a}$$

$$AB + \bar{A}C = (A + C)(\bar{A} + B)$$

Figure 5-11

theorems to the simplification of contact networks. In the first example, the normally-closed A contact is redundant. If the B contact is closed, the network will be closed regardless of the state of relay A : if relay A is unoperated, the $\bar{A}B$ path closes the network; if relay A is operated, the A path closes the network. The second example illustrates the application of

the included-term theorem. The BC path is redundant since there is also a B contact in the AB path and a C contact in the $\bar{A}C$ path. If both of these contacts B and C are closed, then the network will be closed because relay A must be in one state or the other, and either the normally-open A contact or the normally-closed A contact must be closed. In the third example, the transposition theorem is applied to the resultant circuit from the second example. Note that the transposition reintroduces the included path BC .

We shall now go beyond the simplification of Boolean functions and examine further simplifications peculiar to contact networks: transfer contacts, bridge circuits, nonplanar networks, graphical complementation, and multi-output networks.

Transfer Contacts

Relay contact terminology includes the terms *springs*, *contacts*, and *positions*. There are three types of springs as shown in Fig. 5-12. Normally-open contacts are made up of two springs (Fig. 5-13); normally-closed contacts are made up of two springs (Fig.

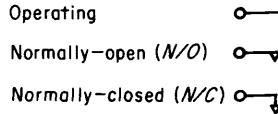


Figure 5-12

5-14). Transfer contacts are made up of a normally-open contact and a normally-closed contact sharing a common operating spring. Thus, they are made up of three springs (Fig. 5-15). Transfer contacts in which the operating spring opens one contact before closing the other contact are called “break-before-make” transfer contacts. With these transfer contacts,



Figure 5-13



Figure 5-14



Figure 5-15

there is a brief period of time during relay operation when both the normally-open and normally-closed contacts are open. Transfer contacts in which the operating spring closes one contact before opening the other contact are called “make-before-break” or “continuity-transfer” contacts. With these transfer contacts, there is a brief period of time during relay operation when both the normally-open and normally-closed contacts are closed.

In a contact network, it is desirable to bring together in an optimum manner normally-open and normally-closed contacts on the same relay to make transfer contacts. A normally-open and a normally-closed contact

that are not combined require four springs in all; if they are combined to make a transfer contact, only three springs are required. An example of this is shown in Fig. 5-16.

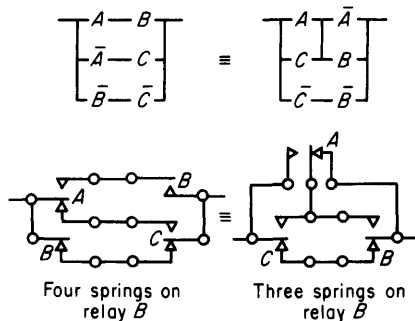


Figure 5-16

Sometimes relays are built up of springs as needed, and it is desirable to minimize the number of springs. In other cases, relays come standardly equipped with a fixed number of transfer contacts. For instance, a particular type of relay might be available in three sizes: 4, 6, or 12 transfer contacts; these are referred to as 4-, 6-, or 12-position relays. In each position, a transfer contact is available. If

a circuit specifies a transfer contact, one position on the relay is required. If a circuit specifies a N/O contact, again one position is required, the normally-closed spring being left unused. If a circuit specifies a N/C contact, one position is required, the normally-open spring being left unused. Thus, a N/O contact and a N/C contact that are not combined require two relay positions in all; if they can be combined into a transfer contact, only one position is required.

Each literal in a Boolean expression corresponds to a contact in the associated series-parallel network. Therefore, the minimization of the Boolean expression leads to a minimum series-parallel contact requirement. Optimizing the number of transfer contacts enables us to minimize the number of springs or positions, whichever is the criterion. The minimization of positions is especially important if it leads to a smaller standard relay being required, since the cost of a relay is a function of its size. For instance, if seven positions on a relay are needed, a 12-position relay could be required. However, if the circuit can be redesigned to require only six positions, then a 6-position relay could suffice.

If, in a relay contact network,

P = the total number of positions

S = the total number of springs

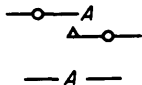
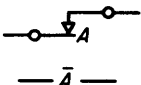
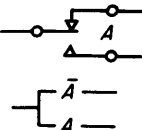
and

C = the total number of contacts

the following relationship exists:

$$P = S - C$$

The following table summarizes some of the relationships discussed.

	N/O Contact	N/C Contact	Transfer Contacts
	 Figure 5-17	 Figure 5-18	 Figure 5-19
Springs	2	2	3
Contacts	1	1	2
Positions	1	1	1

Bridge Circuits

Since, in contact networks, the AND function is implemented by series paths and the OR function by parallel paths, any Boolean expression can be *directly* implemented only by a series-parallel network. Frequently, economy can be achieved by the use of *bridge circuits*. A bridge circuit is one in which there is at least one cross-connecting contact between two series paths (Fig. 5-20).

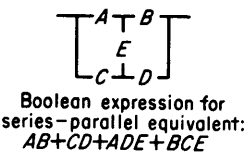


Figure 5-20

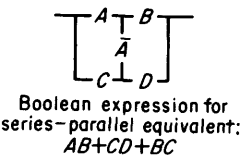


Figure 5-21

A cross-connecting contact in a bridge circuit often conducts current in both directions: for example, the *E* contact in the *A-E-D* and *C-E-B* paths in the circuit above. This is not a necessary requirement, however, as Fig. 5-21 shows.

The circuit in Fig. 5-21 is a bridge even though the cross-connecting \bar{A} contact conducts current in only one direction. Current is prevented from flowing in the other direction because of the *A* and \bar{A} contacts in series.

Nonplanar Networks

Economy is also sometimes achieved by the use of *nonplanar networks* (Fig. 5-22). A nonplanar network is one that *cannot* be drawn on a plane

without crossing lines. The fact that a circuit is drawn with crossovers does not necessarily make it a nonplanar network since it may be possible to redraw the circuit to eliminate the crossovers. Only when it is impossible to draw the circuit without crossovers is the network nonplanar.

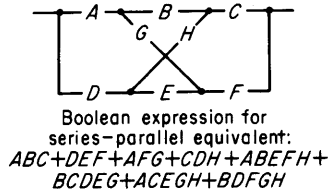


Figure 5-22

Complementation of Contact Networks

If a contact or two-terminal contact network is closed, its complementary network is open, and vice versa. The complement of a two-terminal series-parallel network can be obtained by:

- (1) Changing all *N/O* contacts to *N/C* contacts, and vice versa.
- (2) Changing all series connections to parallel connections, and vice versa.

For example, the complementary network of Fig. 5-23 is Fig. 5-24.

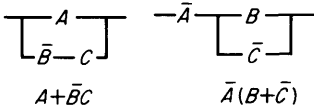
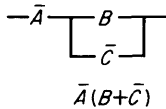


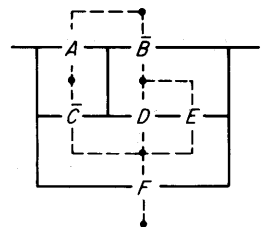
Figure 5-23

Figure 5-24



There is a graphical method, however, for obtaining the complement of any planar two-terminal contact network, including bridges. First, a *mesh* in a contact network will be defined as a closed loop that does not contain any smaller loop.

Thus, in Fig. 5-25, the loop containing contacts *A* and \bar{C} is a mesh; the loop containing \bar{B} , *D*, and *E* is a mesh; and the loop containing contacts \bar{C} , *D*, *E*, and *F* is a mesh. In addition, the area above the network is considered a mesh, as is the area below the network. Thus, the circuit in Fig. 5-25 has five meshes. A point, or *node*, is placed in each mesh as shown. The nodes in adjacent meshes are connected with lines passing through contacts common to both meshes. This is done in all possible ways. These connecting lines must always pass through contacts; they may never “cut a wire.” The input and output terminals are considered as extending to infinity, so that a connection cannot “circle around” an input or output terminal. The progress at this point is shown in Fig. 5-25.



Graphical complementation

Figure 5-25

To obtain the complementary network, the new connections are retained

and the original connections are deleted, and all contacts are complemented, that is, all *N/O* contacts are changed to *N/C* contacts and vice versa. The top and bottom nodes become the input and output terminals of the complementary network. The resultant complementary network is shown in Fig. 5-26.

Figure 5-27 is an example of graphical complementation applied to a bridge circuit.

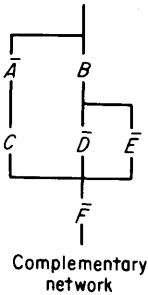


Figure 5-26

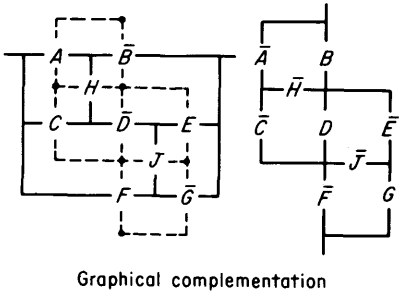


Figure 5-27

The number of contacts required for a complementary network is always the same as that required for the original network, although the spring and position count may differ.

A nonplanar network must be converted to a planar equivalent before graphical complementation can be applied.

Multi-Output Contact Networks

Combining a Network and Its Complement

In designing economical contact networks having more than one output, it is often desirable to combine a network and its complement. Consider first the combination of a simple series circuit containing two contacts, and the complementary circuit made up of the two complementary contacts in parallel.

EXAMPLE:

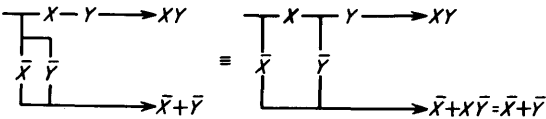


Figure 5-28

To effect the combination, the series circuit is not altered, but the parallel circuit is modified by making use of the theorem

$$X + \bar{X}Y = X + Y$$

in reverse, as the above example illustrates. The use of transfer contacts is optimized since every N/O and N/C contact pair is combined. This procedure can be extended to any series parallel network. Some examples are given in Figs. 5-29 and 5-30.

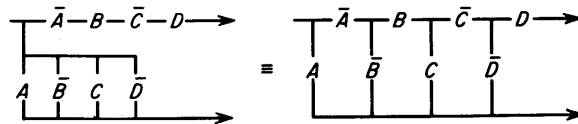


Figure 5-29

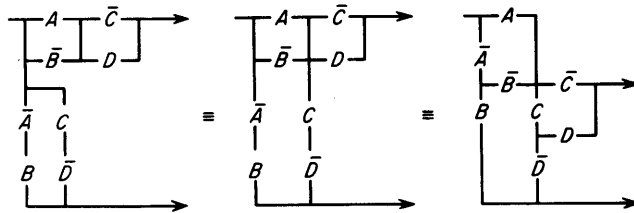


Figure 5-30

The procedure is useful not only in multi-output circuits, but also in a single-output network, a portion of which contains a circuit connected to its complement (Fig. 5-31).

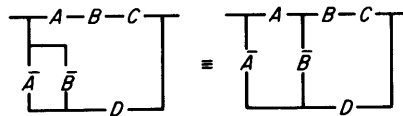


Figure 5-31

Combining Like Contacts in Series Paths

The method of combining like contacts in series paths is intuitively obvious, and bears a relationship to factoring in Boolean algebra.

EXAMPLE:

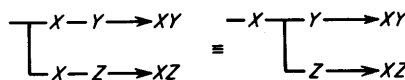


Figure 5-32

Combining Like Contacts in Parallel Paths

In one common method of combining like contacts in parallel paths, the theorem $X + \bar{X}Y = X + Y$ is used in reverse again as illustrated in Figs. 5-33 and 5-34.

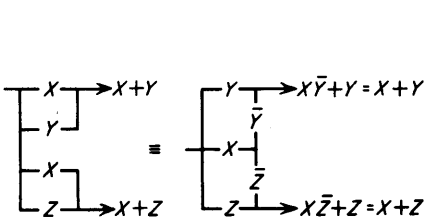


Figure 5-33

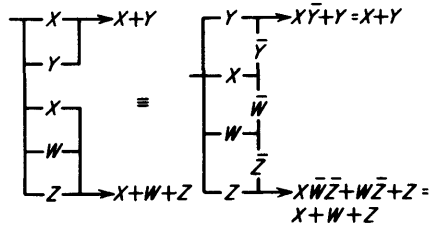


Figure 5-34

If the parallel paths also include, in addition to like contacts, a pair of complementary contacts, that is, one path contains a contact, and the other path contains the complementary contact, another method of combining can be used, as illustrated in Fig. 5-35. The common contact X bridges across the two paths, one of which contains the contact Y , and the other, the complementary contact \bar{Y} .

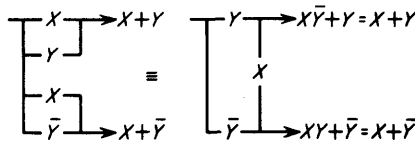


Figure 5-35

This combining by bridging can be used whenever there is at least one contact and its complement in the two paths respectively, the bridging consisting of the circuitry common to both paths. It does not matter what other contacts might be in the two paths.

EXAMPLE:

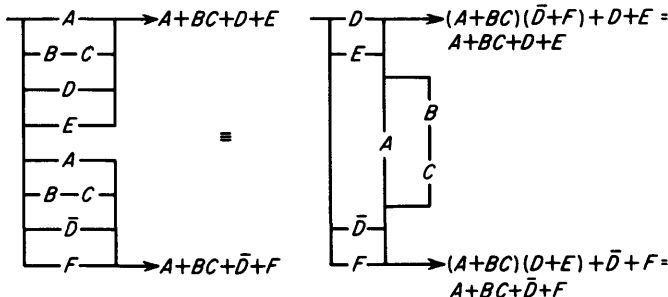


Figure 5-36

Because of the D in one output path, and the \bar{D} in the other path, the circuitry common to both paths, $A + BC$, can be used to bridge across the outputs.

More than one of these simplification procedures can sometimes be applied to a single problem, as shown in Fig. 5-37.

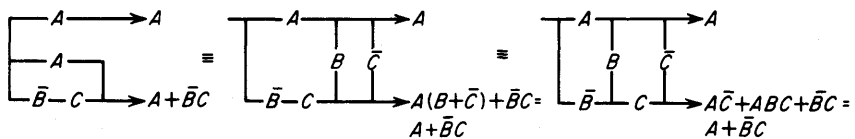


Figure 5-37

PROBLEMS

1. Redraw the following circuit pictorially (Fig. 5-38).

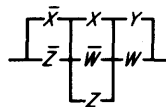


Figure 5-38

- *2. Redraw the following 9-spring circuit pictorially (Fig. 5-39).

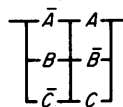


Figure 5-39

3. Design a relay contact circuit to realize the following expression (9 springs). Draw pictorially.

$$ABC + \bar{A}\bar{B}\bar{C}$$

4. Using the graphical method, obtain the complement of the circuit in Fig. 5-40.

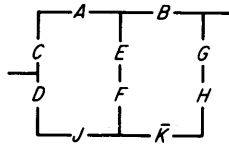


Figure 5-40

5. Using the graphical method, obtain the complement of the circuit in Fig. 5-41.

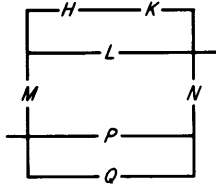


Figure 5-41

- *6. Using the graphical method, obtain the complement of the circuit in Fig. 5-42.

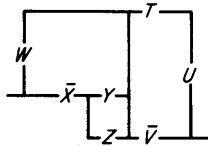


Figure 5-42

- *7. Using the graphical method, obtain the complement of the circuit in Fig. 5-43.

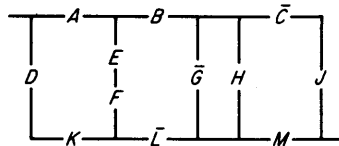


Figure 5-43

8. Two relay circuits, made up of contacts on nine relays, A through J , are required. One circuit is to be closed only for a prescribed 205 combinations of relay states. This circuit is shown in Fig. 5-44. The other circuit is to be closed only for the *other* 307 combinations of relay states. Design this circuit.

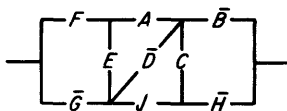


Figure 5-44

9. A two-output relay circuit is required, the expressions for the outputs being:

$$\text{Output 1: } A(\bar{B} + C) + \bar{D}E$$

$$\text{Output 2: } (\bar{A} + B\bar{C})(D + \bar{E})$$

Design the circuit using one transfer contact per relay.

10. Design the following two-output relay circuits, in each case using only one position on each relay.

(a) Output 1: $AB + C$

Output 2: $AD + E$

(b) Output 1: $A + BC$

Output 2: $A + D + E$

(c) Output 1: $A(B + C) + D + E$

Output 2: $A(B + C) + \bar{E} + F$

11. Redesign the following multi-output network (Fig. 5-45), using 10 springs.

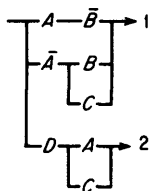


Figure 5-45

6

Tabular Method of Simplification

Optional Combinations

Until now, for a desired circuit function, all of the possible input combinations could be considered as being divided into two groups: one group consisting of those combinations for which a circuit output is desired, and the other group consisting of those combinations for which no output is desired.

For example, suppose a circuit output is specified by:

$$\bar{A}\bar{C} + ABC$$

This expression expands into

$$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC$$

No output is desired for the remaining five possible combinations:

$$\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

which can simplify to

$$\bar{A}C + A\bar{C} + A\bar{B}$$

or

$$\bar{A}C + A\bar{C} + \bar{B}C$$

Note that the expressions for *output* and *no output* are complementary.

Now, all of the possible input combinations will be considered as being divided into *three* groups: one group consisting of those combinations for which an output is desired; another group of those combinations for which no output is desired; and the third group of *optional* combinations.

These optional input combinations arise from two possible conditions:

1. The optional combinations are invalid; that is, they are known never to occur.
2. The optional combinations are "don't care" combinations; that is, we do not care whether or not we get an output if these input combinations occur.

It is not necessary to differentiate between invalid and don't care combinations; they both influence a circuit expression in the same way. If optional combinations are added to a Boolean expression for a circuit output, the expression may become simpler or more complicated. It must therefore be determined which optional combinations to add, and which not to add, in order to achieve optimum simplification.

EXAMPLES:

- (a) Suppose that there are two keys *A* and *B* which operate contacts. It is desired that these keys light a lamp only if key *A* is depressed and key *B* is not depressed. The Boolean expression for lighting the lamp is $A\bar{B}$. Suppose, furthermore, that the keys are mechanically interlocked so that only one can be depressed at a time. It would thus be impossible to depress both keys together, which means that *the combination AB can never occur*. If this optional combination AB is added to the output expression $A\bar{B}$, the expression $A\bar{B} + AB = A$ results. Therefore, instead of requiring a normally-open *A* contact and a normally-closed *B* contact in series to light the lamp, only a normally-open *A* contact is required. It can be seen intuitively that this is true, since if key *A* is depressed, it is not necessary to stipulate that key *B* be not depressed; it cannot be depressed since keys *A* and *B* cannot be depressed together because of the interlock.

In this example, utilization of the optional combination led to simplification. In the next example, it will lead to complication.

- (b) Using the same two interlocked keys, suppose now that the lamp is to light only when neither key is depressed. The Boolean expression for

lighting the lamp is $\bar{A}\bar{B}$. If the optional combination AB is added to the output expression $\bar{A}\bar{B}$, the expression $\bar{A}\bar{B} + AB$ results. Since utilization of the optional combination complicates the expression, rather than simplifies it, it is better not to add the optional combination, but leave the expression in its original form, $\bar{A}\bar{B}$.

In these two examples only one optional combination was involved, and it was not very much work to investigate whether or not its utilization led to simplification.

Now, an example with two optional combinations will be examined. The expression for the output is:

$$AC\bar{D} + BC\bar{D}$$

and the combinations $\bar{A}\bar{B}CD$ and $\bar{A}\bar{B}C\bar{D}$ are optional.

Using neither optional combination, the original expression can be factored, giving

$$(1) \quad (A + B)C\bar{D}$$

Using both optional combinations,

$$(2) \quad AC\bar{D} + BC\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} = AC\bar{D} + BC\bar{D} + \bar{A}\bar{B}C \\ = C(\bar{D} + \bar{A}\bar{B})$$

an expression of the same complexity is obtained.

Using just the optional combination $\bar{A}\bar{B}CD$,

$$(3) \quad AC\bar{D} + BC\bar{D} + \bar{A}\bar{B}CD = C[(A + B)\bar{D} + \bar{A}\bar{B}D]$$

a more complicated expression results.

Finally, using only the optional combination $\bar{A}\bar{B}C\bar{D}$,

$$(4) \quad AC\bar{D} + BC\bar{D} + \bar{A}\bar{B}C\bar{D} = C\bar{D}$$

maximum simplification is achieved.

With two optional combinations, four trials were necessary to determine the optimum solution. In general, with n optional combinations, 2^n such trials are necessary. Obviously, n does not have to be very large before the work involved in this sort of algebraic simplification becomes prohibitive.

The tabular and map methods of simplification, however, handle optional combinations with facility, as will be seen in the following sections.

Tabular Method of Simplification

The tabular method of simplification is based principally on the theorem

$$XY + X\bar{Y} = X$$

X representing one or more variables, and Y representing a single variable.

The first step in the tabular method is to get the expression (to be simplified) in the expanded sum of products form. The preceding theorem is then applied exhaustively to obtain all irreducible terms, that is, terms to which the theorem cannot be further applied.

The theorem is applied first to all possible pairs of terms. Two terms to which the theorem can be applied will reduce to one term which is smaller by one literal. For example,

$$ABC + ABC\bar{C} = AB$$

Next, all terms reduced by one literal are examined to see whether they can be combined further, by the application of the theorem, to reduce to a still smaller term containing two fewer literals than the original terms. This procedure is continued until no further terms can be combined. The resulting irreducible terms are called "prime implicants."

The last step in the method is to select in an optimum manner prime implicants that account for all of the original expanded terms. These prime implicants will form a minimum sum of products.

In the reduction process, the following relationships hold:

n = number of variables

m = number of variables occurring in all possible combinations in 2^m terms

$(n - m)$ = number of variables constant in the 2^m terms

The 2^m terms reduce to a single term defined by the constant $(n - m)$ variables.

EXAMPLE:

$$A\bar{B}\bar{C}\bar{D}\bar{E} + A\bar{B}\bar{C}\bar{D}E + A\bar{B}\bar{C}D\bar{E} + A\bar{B}CDE = A\bar{B}\bar{C}$$

n (the number of variables) = 5,

m (the number of variables occurring in all possible combinations) = 2 (D and E); these combinations occur in $2^m = 2^2 = 4$ terms.

The remaining $(n - m = 3)$ variables are constant in the four terms, and the expression reduces to $A\bar{B}\bar{C}$.

In the tabular method of simplification, the expression to be simplified must first be expanded, if it is not already in expanded form. However, instead of the expanded expression being written in algebraic form, it can be written in tabular form. For example,

$$\bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D}$$

can be written

$$\bar{A}\bar{B}\bar{C}D$$

$$\bar{A}\bar{B}\bar{C}\bar{D}$$

$$\bar{A}B\bar{C}\bar{D}$$

$$A\bar{B}\bar{C}\bar{D}$$

This table can be written in a still more convenient form by simply using A , B , C , and D as columnar headings, and then, in the table, using a 1 to represent an uncomplemented literal and a 0 to represent a complemented literal. The preceding table would then be written

A	B	C	D
0	0	1	1
0	1	0	1
0	1	1	0
1	1	0	0

In the study of the tabular method, the following expression will be used as an example.

$$AB + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + \bar{A}BCD + \bar{A}B\bar{C}\bar{D}$$

The first step is to expand this expression into a table. The AB term will expand into four terms; the last four terms are already in expanded form, and none of these is the same as a term resulting from the expansion of AB . Therefore, the table has eight rows, as shown below.

A	B	C	D
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0

Instead of examining all possible pairs of rows for application of the theorem, the work can be simplified by the following reasoning. For two rows to combine, they must differ in only one column; in one row, that column must contain a 0; in the other row, that column must contain a 1. Thus, a necessary condition for two rows to combine is that one of the rows must contain one more 1 than the other row. If, therefore, the rows are grouped according to the number of 1's per row, and the groups are arranged consecutively according to the number of 1's per row, it is necessary only to compare rows in one group with rows in an adjacent group. The table is thus reordered. Lines are drawn between adjacent groups to aid in identification.

	A	B	C	D
One 1 per row	0	1	0	0
Two 1's per row	1	1	0	0
	1	0	1	0
	1	0	0	1
Three 1's per row	1	1	0	1
	1	1	1	0
	0	1	1	1
Four 1's per row	1	1	1	1

Now we look for rows that combine. The 0100 row is compared with the three rows in the next group. The 0100 and 1100 rows differ in only one column, column A ; therefore, these two rows combine into a row which is written, in a new table, as —100.

A	B	C	D	A	B	C	D
0	1	0	0✓	—	1	0	0
1	1	0	0✓	1	1	0	—
1	0	1	0✓	1	1	—	0
1	0	0	1✓	1	—	1	0
				1	—	0	1
1	1	0	1✓				
1	1	1	0✓	1	1	—	1
0	1	1	1✓	1	1	1	—
				—	1	1	1
1	1	1	1✓				

The entry —100 records that the terms $\bar{A}B\bar{C}\bar{D}$ and $AB\bar{C}\bar{D}$ reduce to $B\bar{C}\bar{D}$. Since the 0100 and 1100 rows are accounted for by the new row —100, they are “checked off,” signifying that they are *not* prime implicants.

Since *all* of the prime implicants must be found, we continue to look for all possible combinations of rows, even with rows that have already been checked off. The 0100 row and the 1010 row are compared and it is found that they differ in more than one column; therefore, they do not combine. The 0100 row and the 1001 row are compared, and it is found that they do not combine either. Lines are drawn between adjacent groups in all tables; therefore, a line is drawn under the —100 row.

The three rows in the second group are now compared with the three

rows in the third group. Rows 1100 and 1101 combine to give 110—; rows 1100 and 1110 combine to give 11—0; 1010 and 1110 combine to give 1—10; and rows 1001 and 1101 combine to give 1—01. All other pairs of rows, one from the second group and one from the third group, differ in more than one column, and thus do not combine.

Next, the three rows in the third group are compared with the 1111 row in the last group. Note that all rows with a single 0 will combine with an all-1 row. Thus, new rows 11—1, 111—, and —111 are obtained. It should also be noted that a row with all 0's would combine with all rows containing a single 1.

In this example, all rows in the original table have combined and have been checked off. If any row had not combined, it would have been a prime implicant.

The next step is to compare the rows in the new table in search for further combinations. Again, for two rows to combine, they must differ in only one column; in one row, that column must contain a 0; in the other row, that column must contain a 1. All other columns must be identical, that is, in all other columns, both rows must contain 0's, both rows must contain 1's, or both rows must contain —'s. Note especially that a — in one row must match with a — in another row.

The —'s speed up the comparison process. For instance, in the only row in the first group of the new table, —100, there is a — in the *A* column. In the next group, none of the four rows have a — in the *A* column. Therefore, it can be seen immediately that —100 does not combine and is a prime implicant. A prime implicant is identified by an asterisk, as shown below.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>		<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
—	1	0	0*		1	1	—	—*
1	1	0	—✓					
1	1	—	0✓					
1	—	1	0*					
1	—	0	1*					
1	1	—	1✓					
1	1	1	—✓					
—	1	1	1*					

Next, the four rows in the second group are compared with the three rows in the third group: 110— combines with 111— to give 11— — in a new

table; 11—0 combines with 11—1 to also give 11— —. This 11— — row is not repeated, but it serves to check off two more rows in the table. 1—10 and 1—01 have —'s in the B column, and since there are no —'s in the B column in any row in the third group, it is immediately seen that these two rows are prime implicants. —111 is also a prime implicant. In the newest table there is only a single row, which obviously cannot combine with any other row, and so it is also a prime implicant.

Now that all prime implicants have been obtained, the last step is to select in an optimum manner prime implicants that account for all of the original expanded terms. To do this, a different kind of table is constructed: there is a column for each of the original terms, and a row for each prime implicant. For each prime implicant, a check mark is placed in the columns of those terms accounted for by that prime implicant. The completed table is shown below.

1100	1101	1110	1111	1010	1001	0111	0100	
✓							✓	—100*
		✓		✓				1—10*
	✓				✓			1—01*
			✓			✓		—111*
✓	✓	✓	✓					11— —

A prime implicant with no —'s will account for only one term; a prime implicant with one — will account for two terms; a prime implicant with two —'s will account for four terms, etc. For instance, the first prime implicant —100 ($B\bar{C}\bar{D}$) accounts for two terms 1100 ($AB\bar{C}\bar{D}$) and 0100 ($\bar{A}B\bar{C}\bar{D}$).

Although there is a formal method for determining the optimum selection of prime implicants, a better understanding of the problem can be gained if it is first looked at intuitively. First, any columns with only a single check mark are noted. A single check mark indicates that there is only one prime implicant that will account for the term in that column; therefore, the prime implicant in that row is required in the final expression—it is an "essential prime implicant."

In the example, the last four columns have only a single check mark. The term 1010 is accounted for only by the prime implicant 1—10; 1001, by the prime implicant 1—01; 0111, by the prime implicant —111; and 0100, by the prime implicant —100. The first four prime implicants are

therefore required. Required prime implicants are identified by an asterisk. The terms are checked off as they are accounted for. Prime implicant —100 accounts for the terms 1100 and 0100; 1—10, for the terms 1110 and 1010; 1—01, for 1101 and 1001; and —111, for 1111 and 0111. At this point, it can be seen that all terms have been accounted for, and the first four prime implicants are the only ones required.

Forming a sum of products with these four prime implicants gives the minimum sum of products equivalent to the original expression

$$B\bar{C}\bar{D} + AC\bar{D} + A\bar{C}D + BCD$$

This particular problem was easy because the accounting for the columns with single check marks effected the solution. In some problems, many or all columns may have many check marks, making the solution of this last step by intuitive methods more difficult. In the next example a formal method of accomplishing this last step will be examined, as well as the method for treating optional combinations.

Optional Combinations with Tabular Method

Any optional combinations are added at the bottom of the original table and, for identification purposes, a line is drawn separating them from the valid combinations. The optional combinations and valid combinations are not differentiated again until after all prime implicants have been obtained; that is, in reordering the table and finding all prime implicants the optional combinations and valid combinations are treated alike.

After the prime implicants have been obtained, the final table is constructed with columns for the valid combinations only, since only the valid combinations must be accounted for. The optional combinations are thus used only for the possible generation of additional prime implicants, or prime implicants with fewer literals.

EXAMPLE:

In the following example there are nine valid combinations and two optional combinations, 0000 and 0100. It is suggested that, for practice, the reader carry out the steps shown without referring to the book, and then check his results. Note that in the final table, there are columns only for the valid combinations. The optional combinations account for the two missing check marks in the third row and the one missing check mark in the fourth row.

Example with Optional Combinations

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
0	0	0	1	0	0	0	0✓	0	0	0	—✓	0	—	0	—*
0	0	1	1					0	—	0	0✓				
0	1	0	1	0	0	0	1✓					—	1	0	—*
1	0	1	0	0	1	0	0✓	0	0	—	1*				
1	0	1	1					0	—	0	1✓	1	—	1	—*
1	1	0	0	0	0	1	1✓	0	1	0	—✓	1	1	—	—*
1	1	0	1	0	1	0	1✓	—	1	0	0✓				
1	1	1	0	1	0	1	0✓								
1	1	1	1	1	1	0	0✓	—	0	1	1*				
								—	1	0	1✓				
0	0	0	0	1	0	1	1✓	1	0	1	—✓				
0	1	0	0	1	1	0	1✓	1	—	1	0✓				
				1	1	1	0✓	1	1	0	—✓				
								1	1	—	0✓				
				1	1	1	1✓								
								1	—	1	1✓				
								1	1	—	1✓				
								1	1	1	—✓				

0001	0011	0101	1010	1011	1100	1101	1110	1111	
✓	✓								00—1 <i>U</i>
	✓			✓					—011 <i>V</i>
✓		✓							0—0— <i>W</i>
		✓			✓	✓			—10— <i>X</i>
			✓	✓			✓	✓	1—1— <i>Y</i>
					✓	✓	✓	✓	11— — <i>Z</i>

Algebraic Solution of Final Table

The formal solution of the final table, interestingly enough, utilizes Boolean algebra. The prime implicants are the variables, and are given letter names; in the example, they are designated *U*, *V*, *W*, *X*, *Y*, and *Z*, respectively.

For each valid combination, a Boolean expression is written indicating which prime implicants can account for it; this expression, by its nature, will be a sum. Thus, in the example, the combination 0001 can be accounted

for by the prime implicants U or W , which is written in Boolean algebra as $(U + W)$; the combination 0011 can be accounted for by the prime implicants U or V ($U + V$); etc.

A product of all these sums is formed, the resulting Boolean expression indicating how all of the combinations in the table can be accounted for. In the example, the product of sums obtained is

$$(U + W)(U + V)(W + X)(Y)(V + Y)(X + Z)(X + Z)(Y + Z)(Y + Z)$$

Some obvious simplification gives

$$(U + W)(U + V)(W + X)(Y)(X + Z)$$

This expression "says" that all the combinations in the table can be accounted for by the prime implicants (U or W) and (U or V) and (W or X) and (Y) and (X or Z).

The product of sums is now multiplied out, with obvious simplifications, to given an *equivalent* expression in the sum of products form:

$$\begin{aligned}(U + W)(U + V)(W + X)(Y)(X + Z) \\ &= (U + VW)(X + WZ)Y \\ &= UXY + UWYZ + VWXY + VWYZ\end{aligned}$$

This sum of products expression logically states the same thing as the previous product of sums expression, except in another way: it states that all the combinations in the table can be accounted for by the prime implicants (U and X and Y) or (U and W and Y and Z) or (V and W and X and Y) or (V and W and Y and Z). In general, the smallest term is selected to account for the table, since it represents the fewest required prime implicants.

In the example, the UXY term is selected because it is the smallest, and the prime implicants

$$U = 00-1 = \bar{A}\bar{B}D$$

$$X = -10- = B\bar{C}$$

$$Y = 1-1- = AC$$

are used to account for the table. If any other term had been selected, four rather than three prime implicants would have been required.

The selected prime implicants are summed to obtain the minimum sum of products, and the solution is

$$\bar{A}\bar{B}D + B\bar{C} + AC$$

This method of solution of the table not only gives *one* minimum sum of products, it gives *all* possible minimums if there are more than one. Furthermore, it gives *all irredundant solutions*, that is, all solutions from which no prime implicant may be removed and still have all output combinations

accounted for. In the preceding example, there are four irredundant solutions.

In the example, examination of the table before applying the algebraic method would show that prime implicant Y is required because the combination 1010 has only a single check mark, which is in the Y row. Prime implicant Y also accounts for the combinations 1011, 1110, and 1111. The algebraic method could be used to account for the remaining five combinations in the table, prime implicant Y being added to the expression obtained. The inspection of a table for columns with single check marks can thus simplify the work involved in solving the table.

In multiplying out the product of sums, use should be made of the simplification theorems whenever possible. Note that since there are no complemented variables involved, only a few of the simplification theorems need be considered. Also, one can be selective in which factors he chooses to multiply out first; if those with the most literals in common are multiplied together first, the process is simplified.

Weighting of Prime Implicants

If there is more than one solution with a minimum number of prime implicants, the solution with the least number of literals is usually desired. As an aid in obtaining the desired solution, each prime implicant can be assigned a "weight" according to the number of literals it contains, and the solution with the smallest weight selected.

In the previous example, the weights would be assigned as follows:

		<i>Weight</i>
U	00—1	3
V	—011	3
W	0—0—	2
X	—10—	2
Y	1—1—	2
Z	11—	2

The terms in the algebraic solution then have weights as follows:

$$UXY + UWYZ + VWXY + VWYZ$$

Weight: 7 9 9 9

With a larger number of variables, it is possible that a solution with the minimum number of prime implicants may contain more literals than a solution containing more prime implicants. If a solution containing a minimum number of literals is desired, this method of weighting would point out which solution to choose.

Simplification of Final Table

If it is sufficient to find *any* minimum solution (rather than all minimums), the final table can often be simplified by the elimination of certain columns and rows, as follows.

1. A column, a , can be eliminated if it has check marks in every row that some other column, b , has. (Column b represents a "tighter" requirement than column a ; that is, if column b is accounted for, column a will be also.)

EXAMPLE:

(a)

	a	b	
	✓	✓	
	✓	✓	
	✓		

Column a can be eliminated because it has check marks in every row that column b has.

(b)

	a	b	
	✓	✓	
	✓	✓	

Either column, a or b , can be eliminated because each has check marks in every row that the other has.

2. A row, z , can be eliminated if some other row, y , has check marks in every column that z has, *and* if the number of literals in the z prime implicant is equal to or greater than the number of literals in the y prime implicant. (The y prime implicant is "stronger" than the z prime implicant in that it accounts, at least, for all columns that z does, and at the same time does not require more literals than z .)

EXAMPLE:

(a)

	✓	✓	✓	011— y
	✓	✓		1—01 z

Row z can be eliminated because row y has check marks in every column that row z has, and the number of literals in the z prime implicant is equal to or greater than the number of literals in the y prime implicant.

(b)

✓		✓		011— y
✓		✓		1—01 z

Either row, y or z , can be eliminated because each has check marks in every column that the other has, and both prime implicants have the same number of literals.

(c)

✓		✓		01— — y
✓		✓		1—01 z

Only row z can be eliminated because even though each row has check marks in every column that the other has, the z prime implicant has a greater number of literals than the y prime implicant.

(d)

✓	✓	✓		011— y
✓	✓			— —01 z

Neither row can be eliminated. Even though row y has check marks in

every column that row z has, the number of literals in the z prime implicant is less than the number of literals in the y prime implicant.

Complementary Approach with Tabular Method

In the complementary approach, a table is made up of those combinations for which *no output* is desired. Any optional combinations are added to the table as before. The table is solved in the usual manner, and the resultant minimum sum of products is complemented using DeMorgan's theorem. Thus the final solution appears in a minimum product of sums form.

Since either the direct or the complementary approach may lead to a solution with fewer prime implicants or fewer literals, it is often desirable to try both solutions, selecting the optimum one. If the number of "output" combinations is large compared to the number of "no-output" combinations, the complementary approach may be used simply to reduce the labor involved in reaching a "good" solution.

If there are no optional combinations, the minimum product of sums obtained using the complementary approach is a true equivalent of the minimum sum of products obtained with the direct approach. However, if there are optional combinations involved, the two expressions *may* not be truly equivalent.

The two expressions will always be equivalent in the sense that if any "output" combination equals 1, both expressions will equal 1, and if any "no-output" combination equals 1, both expressions will equal 0. However, if an optional combination equals 1, the two expressions may or may not be equivalent.

If, with optional combinations, it so happens that the prime implicants used in the direct approach solution and the prime implicants used in the complementary approach solution together account for *all* of the optional combinations, and there is no optional combination accounted for in both approaches, the minimum sum of products obtained in the direct approach and the minimum product of sums obtained in the complementary approach will be equivalent. If any optional combination is not accounted for in either approach, or is accounted for in both approaches, the two resulting expressions will not be equivalent. The expressions obtained with the two approaches may therefore not be logically equivalent; however, lack of equivalence may occur only for optional combinations.

The complementary approach will now be applied to the problem previously solved. Again, it is suggested that, for practice, the reader solve this problem on his own first.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
0	0	1	0	0	0	0	0✓	0	0	—	0✓	0	—	—	0*
0	1	1	0					0	—	0	0✓				
0	1	1	1	0	0	1	0✓	—	0	0	0*				
1	0	0	0	0	1	0	0✓								
1	0	0	1	1	0	0	0✓	0	—	1	0✓				
								0	1	—	0✓				
0	0	0	0	0	1	1	0✓	1	0	0	—*				
0	1	0	0	1	0	0	1✓					0	1	1	—*
				0	1	1	1✓								

0010	0110	0111	1000	1001	
			✓		—000
			✓	✓	100— *
	✓	✓			011— *
✓	✓				0—0 *

This simple table may be solved intuitively, and it is found that the last three prime implicants are required. The resultant sum of products

$$A\bar{B}\bar{C} + \bar{A}BC + \bar{A}\bar{D}$$

complemented using DeMorgan's theorem, gives the minimum product of sums solution

$$(\bar{A} + B + C)(A + \bar{B} + \bar{C})(A + D)$$

In this example it is seen that the complementary approach involved less work because of the fewer combinations involved. However, the minimum product of sums solution contains more literals than the minimum sum of products solution.

The two expressions in this example are not logically equivalent since the optional combination 0100 was accounted for in both approaches—by the prime implicant —10— in the direct approach, and by 0— —0 in the complementary approach. When this optional combination $\bar{A}\bar{B}\bar{C}\bar{D}$ equals 1, the minimum sum of products equals 1, and the minimum product of sums equals 0. For all other possible combinations, the two solutions are equivalent.

The tabular method of simplification can also be used when the original expression is in an expanded product of sums form. The procedure is the same throughout, the selected prime implicants representing a minimum product of sums.

Iterative Method for Obtaining Prime Implicants

In the method described in this chapter, the expanded sum of products form was obtained in order to obtain the prime implicants. If the original sum of products to be minimized is not expanded, it may be desirable to obtain the prime implicants directly, without having to expand first. An iterative method for obtaining the prime implicants from any sum of products will now be described.

The method makes use of Theorem 13 in reverse, and Theorems 3 and 11:

$$13. \quad XY + \bar{X}Z = XY + \bar{X}Z + YZ$$

$$3. \quad X + X = X$$

$$11. \quad X + XY = X$$

The method can be stated very simply. Theorem 13 in reverse is applied systematically to all pairs of terms to obtain all possible included terms, which are added to the expression. The pairing continues as the included terms are added. At the same time, terms are eliminated as Theorems 3 and 11 are applied whenever possible. An included term that can be immediately eliminated by the use of Theorems 3 or 11 is not added. The process is exhaustively continued until no more included terms can be formed, or until the only included terms that can be formed would be immediately eliminated by the use of Theorems 3 or 11. The existing terms at this point comprise all of the prime implicants.

EXAMPLE:

Find all of the prime implicants from the expression

$$\bar{A}\bar{C}D + \bar{A}\bar{B}D + \bar{A}B\bar{C}D + \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$$

The first term, $\bar{A}\bar{C}D$, paired with the other terms, adds the included terms $\bar{B}\bar{C}D$, $B\bar{C}D$, and $\bar{A}\bar{B}D$; $B\bar{C}D$ eliminates $\bar{A}B\bar{C}D$, and $\bar{A}\bar{B}D$ eliminates $\bar{A}\bar{B}\bar{C}D$. We now have

$$\bar{A}\bar{C}D + \bar{A}\bar{B}D + \bar{A}B\bar{C}\bar{D} + \bar{B}\bar{C}D + B\bar{C}D + \bar{A}\bar{B}D$$

The second term, $\bar{A}\bar{B}D$, with the other terms, adds $\bar{A}\bar{C}D$ and $\bar{B}D$; $\bar{B}D$ eliminates $\bar{A}\bar{B}D$, $\bar{B}\bar{C}D$, and $\bar{A}\bar{B}\bar{C}\bar{D}$. The expression at this point is

$$\bar{A}\bar{C}D + \bar{A}B\bar{C}\bar{D} + B\bar{C}D + \bar{A}\bar{C}D + \bar{B}D$$

The next term, $\bar{A}B\bar{C}\bar{D}$, with the other terms, adds $\bar{A}B\bar{C}$, which eliminates $\bar{A}B\bar{C}\bar{D}$. Next, $B\bar{C}D$, with the other terms, adds $\bar{C}D$, which eliminates $\bar{A}\bar{C}D$, $B\bar{C}D$, and $\bar{A}\bar{C}D$. The expression is now

$$\bar{B}D + \bar{A}B\bar{C} + \bar{C}D$$

which comprises all of the prime implicants, since there are no more included terms that cannot be immediately eliminated.

This process can, of course, also be carried out in tabular form.

Multi-Output Networks

Multiple-output networks may sometimes be able to share logic blocks or contacts in common. For example, if the expression for output-1 is $A\bar{B} + CD$, and the expression for output-2 is $CD + \bar{C}\bar{D}$, the CD term can be shared by both circuits, as shown in Fig. 6-1.

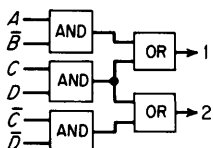


Figure 6-1

The determination of terms that can be shared is not always so obvious. For example, the expressions 1: $\bar{A}BD + \bar{A}\bar{C}\bar{D}$ and 2: $\bar{A}BC + ACD + \bar{B}\bar{C}\bar{D}$ (a total of 5 terms and 15 literals) have no term in common; however, an equivalent expression for output-2 is $\bar{A}\bar{C}\bar{D} + BCD + A\bar{B}\bar{C}$, which has the term $\bar{A}\bar{C}\bar{D}$ in common with the output-1 expression (a total now of 4 terms and 12 literals).

Furthermore, a possible common term may not be a prime implicant! Thus, even an examination of all possible equivalent minimum, or, for that matter, irredundant sums of products may not show up possible terms that can be shared to give an optimum multi-output network. As an example, the expressions 1: $\bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}$ and 2: $AC + \bar{B}\bar{D} + BC$ (6 terms, 12 literals) have no terms in common, but the equivalent expressions 1: $\bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}BC$ and 2: $AC + \bar{B}\bar{D} + \bar{A}BC$ have the non-prime implicant term $\bar{A}BC$ in common (5 terms, 11 literals).

In this section, the tabular method will be extended to multi-output networks. *Multiple-output prime implicants* are obtained from which is selected a set of sums of products or products of sums that is minimum in an overall sense. The method will be illustrated by an example.

The multi-output tables have both input and output columns. All input combinations for which there is at least one output *on* are listed; the outputs that are *on* for each of these input combinations are recorded by a check mark in the appropriate output column. The rows are ordered in the usual manner.

A	B	C	D	1	2	3
0	0	0	1		✓	✓
0	1	0	1	✓		
0	1	1	0			✓
0	1	1	1	✓	✓	✓
1	0	0	0	✓		✓
1	0	0	1	✓		✓
1	0	1	0	✓		✓
1	0	1	1	✓	✓	✓
1	1	0	1	✓		
1	1	1	1		✓	

A	B	C	D	1	2	3
0	0	0	1		✓	✓
1	0	0	0	✓		✓
0	1	0	1	✓		
0	1	1	0			✓
1	0	0	1	✓		✓
1	0	1	0	✓		✓
0	1	1	1	✓	✓	✓
1	0	1	1	✓	✓	✓
1	1	0	1	✓		
1	1	1	1		✓	

The combining of the rows is necessarily modified as follows:

- Only rows with at least one *on* output in common can be combined.
- In the resulting row, only the *on* outputs that were common are checked.
- A combining row is "checked off" (signifying that it is not a prime implicant) only if the resulting row accounts for *all* of its *on* outputs.

For some specific examples, note in the table that rows 0001 and 0101

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
0	0	0	1		✓	✓
0	1	0	1	✓		

cannot be combined because there are no *on* outputs in common.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
0	0	0	1		✓	✓
1	0	0	1	✓		✓

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
—	0	0	1			✓

Rows 0001 and 1001, with *on* output-3 in common, can combine to give the row —001, in which only output-3 is checked. Neither of the combining rows can be checked off, since the resulting row doesn't account for all of the *on* outputs in either case.

Rows 1000 and 1001, with *on* output-1 and *on* output-3

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
1	0	0	0	✓		✓
1	0	0	1	✓		✓

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
1	0	0	—	✓		✓

in common, combine to give the row 100—, in which both output-1 and output-3 are checked. Both combining rows are checked off, since the resulting row accounts for all of the *on* outputs in both cases.

Rows 0101 and 0111, with *on* output-1 in common, combine to give

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
0	1	0	1	✓		✓
0	1	1	1	✓	✓	✓

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
0	1	—	1	✓		

the row 01—1, in which output-1 is checked. The resulting row accounts for all of the *on* outputs in row 0101, but not for all of those in row 0111; therefore, only row 0101 can be checked off. The complete tabulation follows.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3	
0	0	0	1		✓	✓	*
1	0	0	0	✓		✓	✓
0	1	0	1	✓			✓
0	1	1	0			✓	✓
1	0	0	1	✓		✓	✓
1	0	1	0	✓		✓	✓
0	1	1	1	✓	✓	✓	*
1	0	1	1	✓	✓	✓	*
1	1	0	1	✓			✓
1	1	1	1		✓		✓

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3	
—	0	0	1		✓		*
1	0	0	—	✓		✓	✓
1	0	—	0	✓		✓	✓
0	1	—	1	✓			*
—	1	0	1	✓			*
0	1	1	—			✓	*
1	0	—	1	✓		✓	✓
1	—	0	1	✓			*
1	0	1	—	✓		✓	✓
—	1	1	1		✓		*
1	—	1	1		✓		*

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3	
1	0	—	—	✓		✓	*

The multi-output prime implicants having been obtained, we now construct the final table. A column is required for each input-output combination.

1										2				3						
0101	0111	1000	1001	1010	1011	1101	0001	0111	1011	1111	0001	0110	0111	1000	1001	1010	1011	1	2	3
							✓				✓							0001	✓	✓
	✓							✓					✓					0111	✓	✓
				✓					✓								✓	1011	✓	✓
											✓				✓			—001		✓
✓	✓					✓												01—1	✓	
✓																		—101	✓	
												✓	✓					011—		✓
			✓			✓												1—01	✓	
								✓		✓								—111		✓
									✓	✓								1—11	✓	*
		✓	✓	✓	✓									✓	✓	✓	✓	10—	✓	*

For each prime implicant, check marks are placed only in columns corresponding to that prime implicant's *on* outputs. Thus, for example, for prime implicant —001, only output-3 is pertinent; therefore, check marks are placed only in the output-3 0001 and 1001 columns. Check marks are *not* placed in the output-1 1001 or output-2 0001 columns.

The completed table, treated as a whole (i.e., not broken down by output) is solved in the normal manner. The required prime implicants are marked with an asterisk.

One last step must now be made. A selected prime implicant may not be required by *all* of its *on* outputs. Therefore, a check must be made for each output, to determine if any of its corresponding prime implicants is redundant as far as that particular output is concerned. One case of such redundancy exists in the present example, and relates to output-3. The relevant portion of the table is extracted for instructional purposes. Note that only

3											
	0001	0110	0111	1000	1001	1010	1011		1	2	3
	✓							0001		✓	✓
			✓					0111	✓	✓	✓
		✓	✓					011—			✓
				✓	✓	✓	✓	10—	✓		✓

the selected prime implicants pertinent to output-3 are considered.

Examination of the table shows that, with regard to output-3, prime implicant 0111 is redundant.

Note that this last step does not affect the total number of terms or literals involved; it may, however, reduce the number of OR logic block inputs or, in the event that an expression is reduced to a single term, eliminate an OR logic block.

The optimum set of expressions for the multi-output network is:

- (1) $\bar{A}BCD + B\bar{C}D + A\bar{B}$
- (2) $\bar{A}\bar{B}\bar{C}D + \bar{A}BCD + ACD$
- (3) $\bar{A}\bar{B}\bar{C}D + \bar{A}BC + A\bar{B}$

The resulting network is shown in Fig. 6-2.

If there are any optional input combinations, they are added to the original table in the usual manner; for these combinations, it should be assumed that all outputs are *on*. Also, for a valid input combination, *some* outputs may be optional; it should be assumed that these outputs are *on*

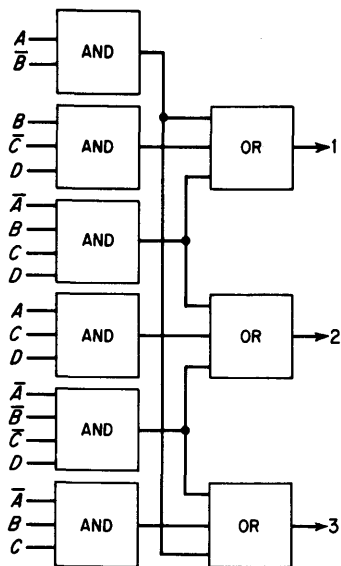


Figure 6-2

also. These optional input-output combinations are used only for obtaining the prime implicants. In the final table, there are columns for only the valid input-output combinations.

EXAMPLE:

The input combination 0010 can never occur; for the combination 0100, we don't care what any of the outputs are; for the combination 1100, output-1 must be on, output-2 must be off, and output-3 can never occur; for the combination 1110, output-1 and output-3 must be off, and we don't care what output-2 is.

These combinations are added to the original table, as follows.

A	B	C	D	1	2	3
0	0	1	0	✓	✓	✓
0	1	0	0	✓	✓	✓
1	1	0	0	✓		✓
1	1	1	0		✓	

In the final table, the only one of these input-output combinations for which there will be a column (the only one that must be accounted for) is output-1 1100.

PROBLEMS

1. The following table represents the expanded sum of products. Obtain the minimum sum of products, using the tabular method.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
0	0	0	1
0	0	1	0
0	0	1	1
1	0	0	1
1	1	0	0
1	1	0	1
1	1	1	0
<hr/>			
0	0	0	0
0	1	1	1
1	0	1	0

2. In the preceding problem, use the complementary approach and compare results.
3. The following table represents the expanded sum of products. The combination $A\bar{B}C\bar{D}$ is optional. Using the tabular method, find all prime implicants, and express algebraically.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
0	0	0	0
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0
<hr/>			
1	0	1	0

4. Given the output combinations and prime implicants below, and using the algebraic method of solution, determine:
- (a) the number of irredundant solutions
 - (b) the minimum-term sum of products
 - (c) the minimum-literal sum of products

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>					
0	0	1	0	0	0	0	0	1	0
					0	1	0	0	1
					0	0	0	1	1
									00—0 <i>U</i>
									0—0—1 <i>V</i>
									000— <i>W</i>
									—1——1 <i>X</i>
									— — —1— <i>Y</i>
									— — —1—0 <i>Z</i>

- *5. The circuit shown in Fig. 6-3 was designed without the knowledge that the three input combinations $\bar{A}\bar{B}\bar{C}\bar{D}$, $\bar{A}B\bar{C}D$, and $A\bar{B}C\bar{D}$ were invalid. Using the tabular method, redesign the circuit, taking advantage of these optional combinations.

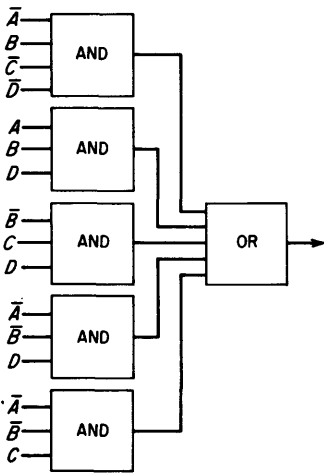


Figure 6-3

6. Design an optimum multi-output network for the requirements in the following table.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	1	2	3
0	0	0	0	✓	✓	
0	0	0	1		✓	
0	0	1	0	✓	✓	✓
0	0	1	1			✓
0	1	0	0	✓		✓
0	1	0	1		✓	
0	1	1	1	✓	✓	✓
1	0	0	0	✓	✓	
1	0	1	0	✓	✓	
1	1	1	1	✓		

7

Map Method of Simplification

The underlying principles of the map method of simplification are basically the same as those for the tabular method. Maps are easy to use because the expression to be simplified is automatically expanded as it is entered on the map, and the prime implicants can be identified by the visual recognition of certain basic patterns. However, some practice is required before the user can feel confident in the use of maps, particularly when the number of variables becomes large.

A map for n variables contains 2^n squares, there being a square on the map for every possible input combination. A 1 is placed in each square representing a combination for which an output is desired; a 0 is placed in each square representing a combination for which no output is desired; and a — is placed in each square representing an optional combination. Often, to reduce the writing, the 0's are omitted, and a blank square is understood to represent a no-output combination.

Figure 7-1 shows two forms of a two-variable map. Although two-variable maps are seldom if ever actually used for simplification, an analysis

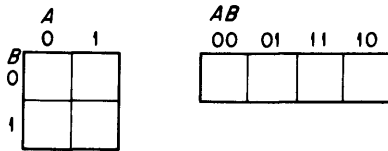


Figure 7-1

of some examples using the two-variable map in both forms will aid in an understanding of the fundamental principles involved.

In entering a map with an expanded term, a 1 is placed in the square of the map corresponding to that expanded

term. The entry for $\bar{A}B$ is shown in Fig. 7-2.

In entering the map with a term that is not expanded, a 1 is placed in all squares defined by that term. The entries for the term B are shown in Fig. 7-3.

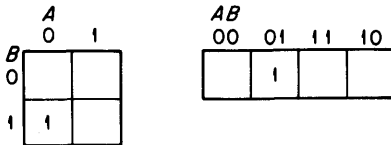


Figure 7-2

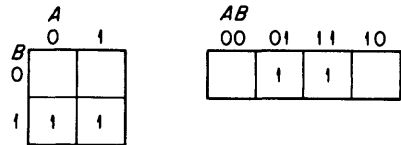


Figure 7-3

Note that if B had first been expanded into $\bar{A}B + AB$, the same two entries would have been made. Thus, B was automatically expanded as it was entered on the map.

In "reading" a map, two 1-squares that are adjacent either horizontally or vertically can be grouped. Larger numbers of 1-squares can also be grouped, the number of squares in a group always being some power of 2; however, for the time being, only groups of two will be considered. The variables that are constant for the group of 1-squares define the group. Thus, the map in Fig. 7-3 is read as B .

The map in Fig. 7-3 might have been entered with $\bar{A}B + AB$. With the map entered, it is observed that two 1-squares are adjacent. This group of two 1-squares is defined by B . Therefore, the term B is read from the map, accomplishing the simplification.

The four possible groups of two 1-squares in a two-variable map are shown in Fig. 7-4.

Note the "reflected" binary ordering¹ of the variables in the right-hand map in each case. With this ordering, any two adjacent squares will differ in only one variable. Thus, all possible groups can be formed by two adjacent 1-squares. The fourth case, that for \bar{B} , warrants special attention. Note that the left-hand square 00 differs from the right-hand square 10 in only one variable. These two squares are considered adjacent in the same sense that the others are adjacent. The adjacency of the two end squares

¹See Chapters 11 and 12.

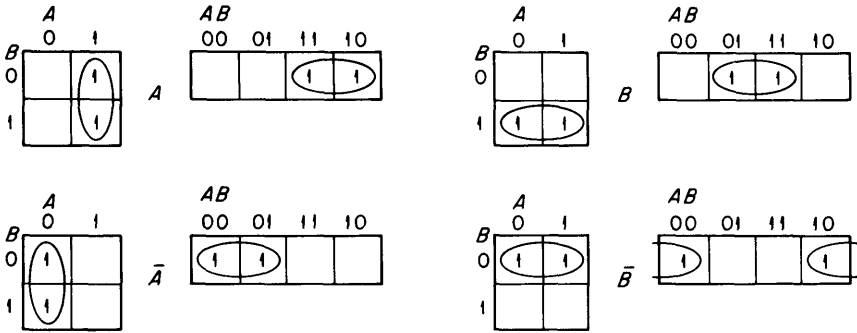


Figure 7-4

may be better appreciated if the map is pictured as rolled into a cylinder, with the right-hand edge touching the left-hand edge.

While in a two-variable map it is not necessary to get involved in this edge-to-edge wrap-around—the square array could have been used instead—this concept has been purposely introduced with the simple two-variable map because it is used in maps of more variables.

If, instead of the reflected binary ordering, a straight binary ordering² had been used, the four previous maps would have looked like Fig. 7-5. Note that the nice relationship of groups always occupying adjacent squares no longer holds. For this reason, the reflected binary ordering is usually used.

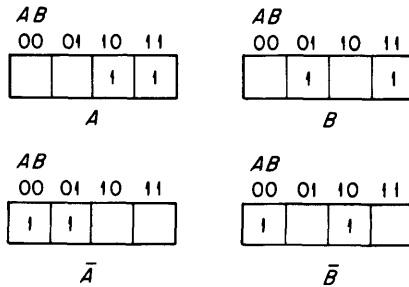


Figure 7-5

In reading a map, every 1-square must be accounted for at least once, although a 1-square may be used in as many groups as desired. Also, a group should be as large as possible, that is, a 1-square should not be accounted for by itself if it can be accounted for in a group of two 1-squares; a group of two 1-squares should not be made if the 1-squares can be included in a group of four; etc. These “largest” groups correspond to prime implicants. All 1-squares should be accounted for in the minimum number of groups, and the resulting expression read from the map will be a minimum sum of products.

²See Chapters 11 and 12.

EXAMPLE:

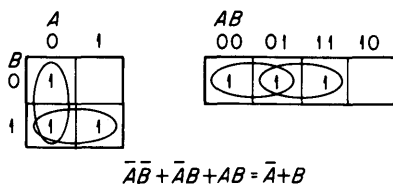


Figure 7-6

In this example, there are two groups of two 1-squares each, one group defined by \bar{A} and the other group by B (Fig. 7-6). Note that the combination $\bar{A}B$ was used in both groups.

EXAMPLES:

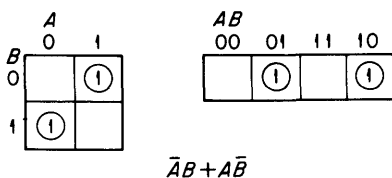


Figure 7-7

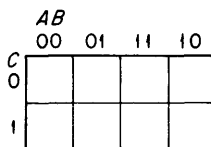


Figure 7-8

In the example of Fig. 7-7 there is no simplification possible; the two 1-squares are not adjacent horizontally or vertically, and $\bar{A}B + A\bar{B}$ is a minimum expression. The construction of a three-variable map is shown in Fig. 7-8. Figure 7-9 shows, for study, some familiar examples of groups of two 1-squares on three-variable maps.

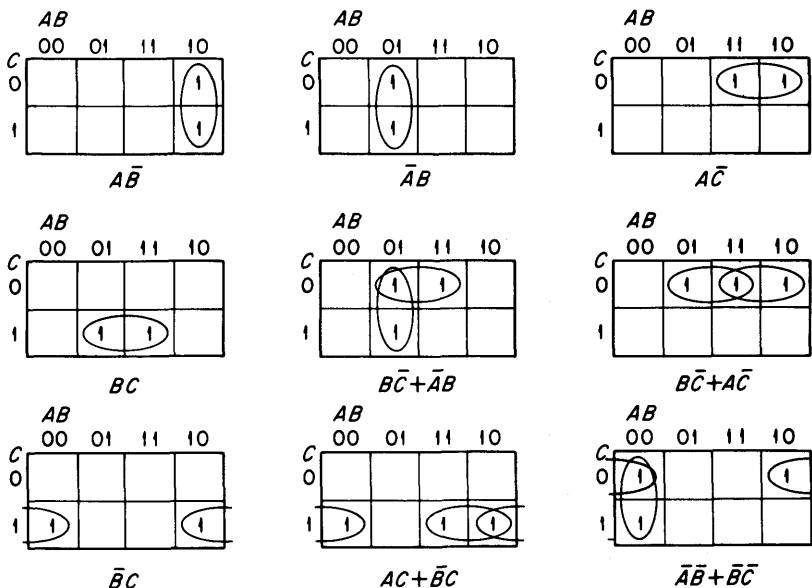


Figure 7-9

Groups of four 1-squares may occur either in a straight line array or in a square array, as shown in Fig. 7-10. Note again the concept of the edge-to-edge wrap-around in the \bar{B} example of Fig. 7-10.

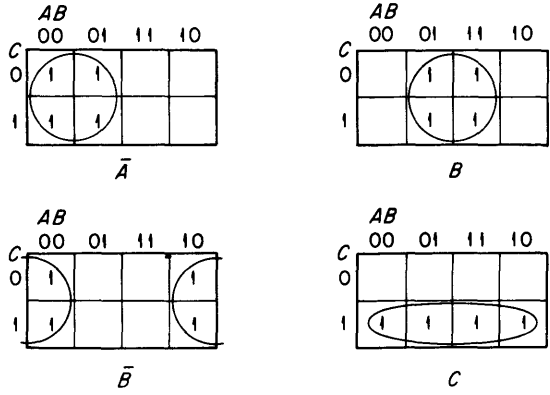


Figure 7-10

Some additional examples are given in Fig. 7-11 to 7-14.

EXAMPLES:

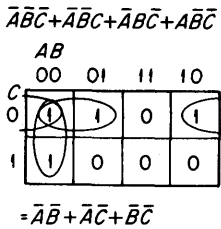


Figure 7-11

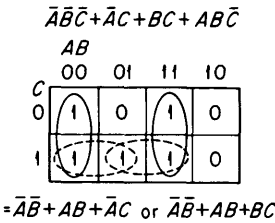


Figure 7-12

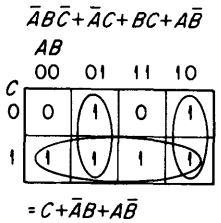


Figure 7-13

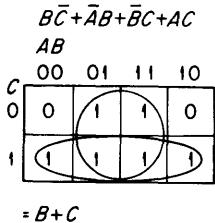


Figure 7-14

In Fig. 7-12, there are two equally good solutions; the $\bar{A}BC$ square can be accounted for by either $\bar{A}C$ or BC .

Note, in all of these examples, that all 1-squares have been accounted for at least once, and that all groups are as large as possible.

Complementary Approach with Map Method

As in the tabular method, it is possible with maps to use the complementary approach. In the preceding four examples, 0's have purposely been entered on the maps in preparation for the discussion of the complementary approach.

In the complementary approach, the 0-squares, rather than the 1-squares, are grouped. Since the resultant sum of products is the complement of the desired expression, this sum of products is complemented using DeMorgan's theorem. The final expression is thus in a minimum product of sums form.

The complementary approach to the preceding four examples is shown in Fig. 7-15.

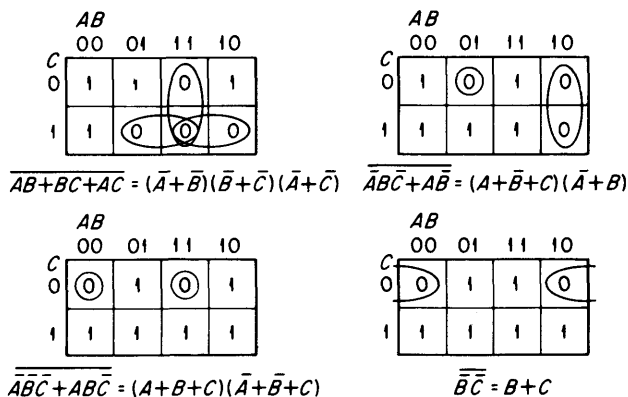


Figure 7-15

Comparison of the equivalent results of both approaches shows that in Example 1, the two solutions are equally optimum; in Example 2, there are fewer literals with the complementary approach; in Example 3, there are fewer literals with the direct approach; and in Example 4, the solutions are identical.

In the complementary approach, the minimum product of sums can be read directly from the map by the mental application of DeMorgan's theorem during the process of reading. For example, in Example 2, instead of the 010 entry being read as the product $\bar{A}\bar{B}\bar{C}$, and later complemented, it can be read directly as the sum $(A + \bar{B} + C)$ by the mental complementation of the variables as the map is read.

Maps are convenient for converting an expression from the sum of products form to the product of sums, or vice versa. For instance, a product

of sums can be entered on a map, with 1's, and a sum of products read from the map by grouping the 0's.

"Method of Attack"

A good general approach in reading the optimum solution from a map is to account first for all 1-squares that can be grouped in only one best way, leaving until last those in which a choice is involved. Look first for any 1-squares that do not combine with any others; these entries must be accounted for by themselves. Next, look for any 1-squares that combine with only one other 1-square; such groups of two should be accounted for next. If a 1-square combines with exactly two other squares, look to see if there is a fourth 1-square that completes a group of four. If there is, the four entries should be accounted for as a group; if not, then there is a choice involved as to which of the two groups of two to choose, and such decisions should be left until last. And so forth. Remaining 1-squares should be combined into the fewest possible groups.

The following simple example illustrates the approach suggested. This example is of interest also because it illustrates the included term theorem.

EXAMPLE:

In this example (Fig. 7-16), there is only one best way to account for the entry $\bar{A}\bar{B}\bar{C}$: with the group $\bar{A}\bar{B}$; and there is only one best way to account for the entry ABC : with the group BC . These two groups account for all entries, and therefore the solution is $\bar{A}\bar{B} + BC$. Note that the entries $\bar{A}\bar{B}\bar{C}$ and $\bar{A}\bar{B}C$ each can combine in two ways, and therefore consideration of these entries is deferred.

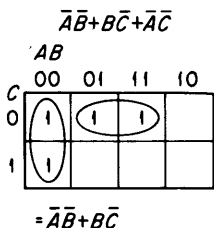


Figure 7-16

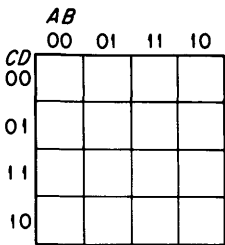


Figure 7-17

A four-variable map is shown in Fig. 7-17. Note the reflected ordering in both the horizontal and vertical directions. In four-variable maps, not only are the left and right edges adjacent, but the top and bottom edges are also adjacent. A mental picture of this left-right top-bottom wrap-around

can be formed if one considers the map as rolled into a cylinder with the left and right edges touching, and then the cylinder rolled into a torus with the top and bottom edges touching.

Figure 7-18 shows a few examples of groups involving these wrap-around adjacencies.

Figure 7-19 shows two examples of groups of eight 1-squares.

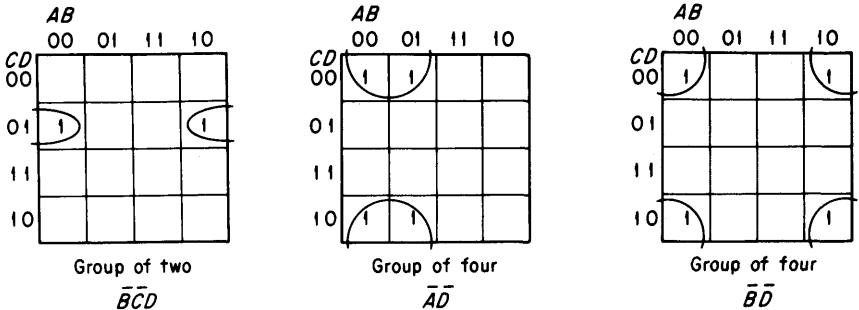


Figure 7-18

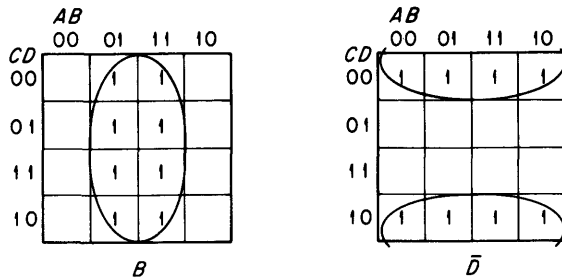


Figure 7-19

Groups should always be as large as possible, that is, every group should correspond to a prime implicant. However, a group should not be made just because it is large. The following example illustrates this important point.

EXAMPLE:

In Fig. 7-20, the group $\bar{A}\bar{C}$ looks very attractive because it is the only group of four on the map. However, all of the entries in this group can

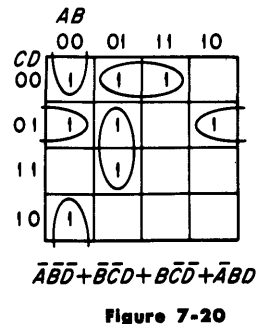


Figure 7-20

combine in more than one way and it is best to consider first those entries that cannot combine in more than one way. Study of the map reveals that each of the other four 1-squares combines with only one other 1-square. When these four groups of two have been made, it is found that every 1-square on the map has been accounted for; thus, the term $\bar{A}\bar{C}$ is redundant.

Figure 7-21 is given for study in both entering and reading a map. The groups are numbered to correspond to the terms from which the map was entered. The map is repeated in Fig. 7-22 showing the groups that are read. These groups are numbered to correspond to the terms in the final expression.

Another comparison of the straight binary ordering and the reflected binary ordering, this time in four-variable maps, is shown in Fig. 7-23.

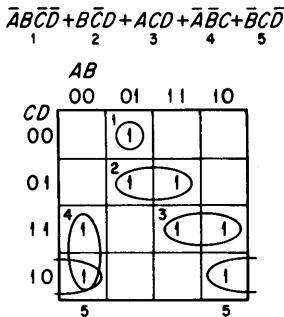


Figure 7-21

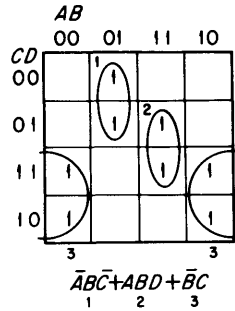


Figure 7-22

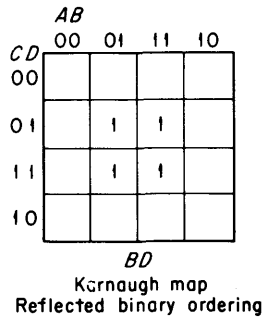
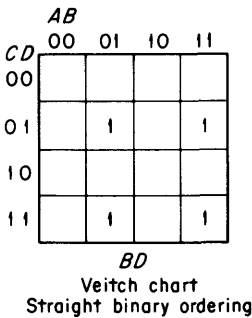


Figure 7-23

The first application of this type of graphical approach to simplification is accredited to E. W. Veitch. The Veitch chart used the straight binary ordering shown on the left. M. Karnaugh modified the Veitch chart, using the reflected binary ordering shown on the right. The resulting improvement is that, in the Karnaugh map, all groups are adjacent rather than some

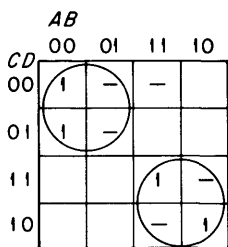
of them being scattered as in the Veitch chart. An alternative method of labeling the Karnaugh map is shown in Fig. 7-24.

Optional Combinations with Map Method

Optional combinations are entered on a map by placing —'s in the corresponding squares. These optional entries may be used in obtaining fewer and/or larger groups. Only the 1-squares *must* be accounted for.

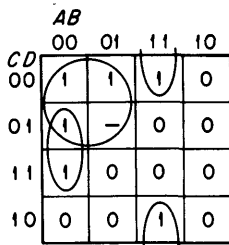
In Fig. 7-25, optional combinations are used to advantage. The optional entries $\bar{A}\bar{B}\bar{C}\bar{D}$ and $\bar{A}\bar{B}\bar{C}D$ are used with the 1-squares $\bar{A}\bar{B}\bar{C}\bar{D}$ and $\bar{A}\bar{B}\bar{C}D$ to give the group $\bar{A}\bar{C}$; the optional entries $\bar{A}\bar{B}CD$ and $\bar{A}BCD$ are used with the 1-squares $\bar{A}\bar{B}CD$ and $\bar{A}BCD$ to give the group $\bar{A}C$. The optional entry $\bar{A}\bar{B}\bar{C}\bar{D}$ is not used.

Optional combinations can be used in both the direct and complementary approach, as shown in Fig. 7-26. Note that the direct approach results in eight literals while the complementary approach results in seven literals. Note also that the two resultant expressions are not true equivalents because the optional combination was used in a group in both approaches.



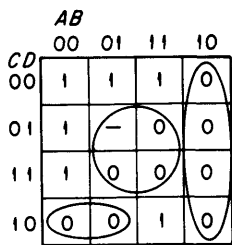
$$\bar{A}\bar{C} + AC$$

Figure 7-25



$$\bar{A}\bar{C} + \bar{A}B\bar{D} + \bar{A}\bar{B}D$$

Figure 7-26



$$(\bar{B} + \bar{D})(\bar{A} + B)(\bar{A} + \bar{C} + D)$$

Maps of More than Four Variables

There are several ways of drawing maps of more than four variables. When three or more variables are involved in *one dimension*, adjacencies are no longer preserved and new patterns must be recognized, as shown in the five-variable map of Fig. 7-27. In addition to the adjacencies already

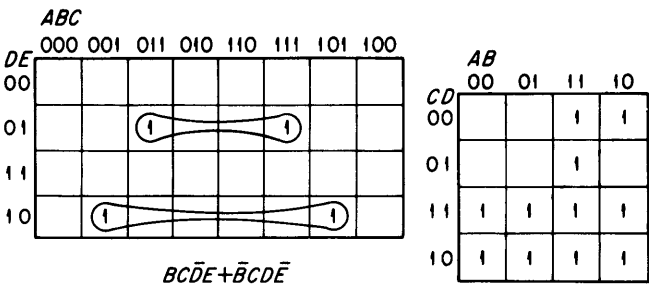


Figure 7-27

discussed, squares equidistant from the vertical center line are considered adjacent.

A more general approach, that can be extended to any number of variables, is shown in the five-variable map of Fig. 7-28. This five-variable map is made up of two four-variable maps drawn side by side. Groups are formed as before except that, in addition to the adjacencies already discussed, corresponding squares on the two maps are considered adjacent. One may picture this adjacency by considering the right-hand map as being situated directly behind the left-hand map, making a three-dimensional map four squares across by four squares down by two squares deep.

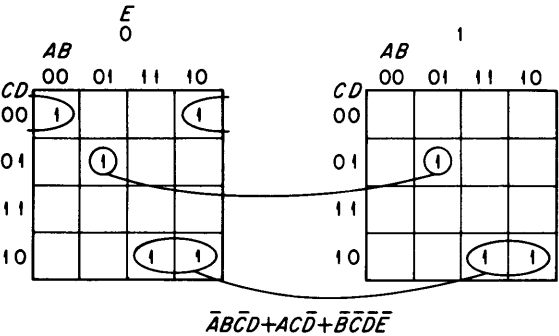


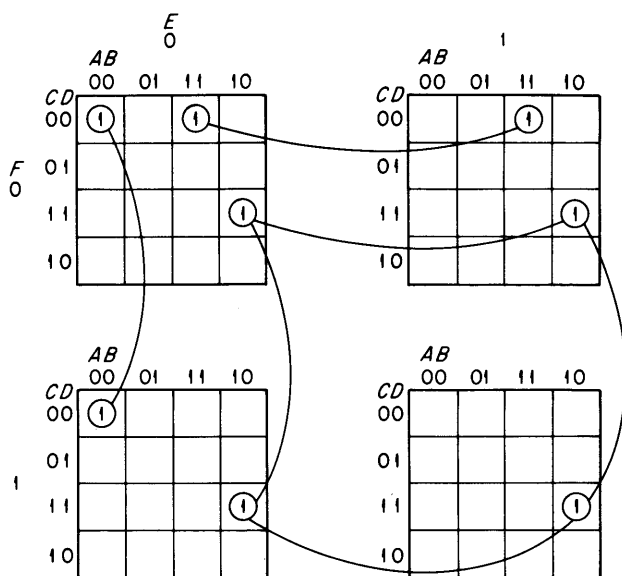
Figure 7-28

In the preceding example, the $\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$ entry on the left-hand map is adjacent to the $\bar{A}\bar{B}\bar{C}DE$ entry on the right-hand map, giving the group $\bar{A}\bar{B}\bar{C}D$. The $AC\bar{D}$ group is also made up of 1-squares from both maps. The $\bar{B}\bar{C}\bar{D}\bar{E}$ group is made up of 1-squares from the \bar{E} map only.

A six-variable map is made up of four four-variable maps drawn in a square array. In addition to the groups that can be formed on any one four-variable map, groups can also be made from corresponding entries on maps horizontally or vertically adjacent.

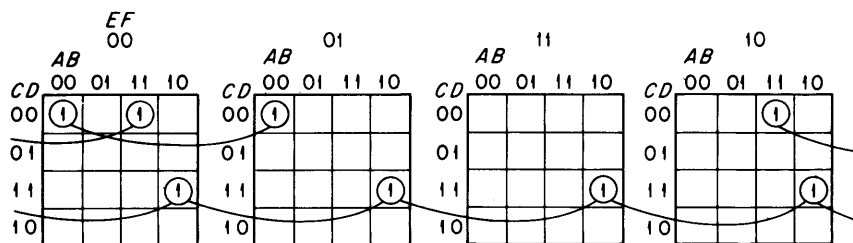
In the map of Fig. 7-29, the group $\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$ comes from corresponding entries on the two left-hand maps; the group $ABC\bar{D}\bar{E}$ comes from corresponding entries on the two upper maps; the term $A\bar{B}CD$ comes from corresponding entries on all four maps.

A seven-variable map is made by placing two six-variable maps side by side. In addition to the groups that can be made on each six-variable map, groups can also be formed from corresponding entries on the two maps. An eight-variable map is made by placing four six-variable maps in a square array; a nine-variable map is made by placing two eight-variable maps side by side; a ten-variable map is made by placing four eight-variable maps in a square array; and so on.



$$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E} + ABC\bar{D}\bar{E} + A\bar{B}CD$$

Figure 7-29



$$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E} + ABC\bar{D}\bar{E} + A\bar{B}CD$$

Figure 7-30

Sometimes a six-variable map is drawn as in Fig. 7-30. The four four-variable maps can be pictured as being placed one behind the other, the left-hand map on top and the right-hand map on the bottom, forming a cube four squares across by four squares down by four squares deep. In this cube, the left-hand and right-hand faces are considered adjacent, the front and back faces are considered adjacent, and the top and bottom faces are considered adjacent.

Summary

Following is a summary of some pertinent points regarding the map method of simplification.

There is a square on the map for every possible combination of variables.

A 1 is placed in each square representing a combination for which an output is desired; a 0 is placed in each square representing a combination for which no output is desired; a — is placed in each square representing an optional combination.

Each 1-square *must* be considered at least once. Each 1-square *may* be considered as often as desired.

Generally, all 1-squares should be accounted for in the minimum number of groups.

Each group should be as large as possible, that is, each group should correspond to a prime implicant.

The number of squares in a group must always be some power of two. In a group of 2^m squares, m variables will occur in all possible combinations. If the total number of variables is n , then $(n - m)$ variables will be constant in these 2^m squares, and these $(n - m)$ variables will define the group.

If the groups are made in an optimum manner, the expression read from the map will be a minimum sum of products.

The complementary approach may be used, in which case the 0-squares rather than the 1-squares are grouped. The groups are complemented and a minimum product of sums is obtained.

Optional combinations may be used to obtain fewer and/or larger groups.

The map method, like the tabular method, can also be directly adapted to the simplification of expressions in the product of sums form, each 1-square representing an expanded sum, and the selected groups representing a minimum product of sums.

Factoring on the Map

The only algebraic simplification possible to perform on the minimum expressions read from the map is factoring. Factoring can also be done directly on the map, as illustrated in Fig. 7-31. Note that there is "almost"

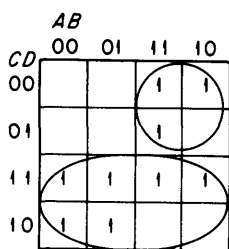


Figure 7-31

a group $A\bar{C}$; there is one 1-square missing from this potential group of four, the $A\bar{B}\bar{C}D$ square. This group can thus be described as " $A\bar{C}$ but not $A\bar{B}\bar{C}D$," which can be written algebraically as $A\bar{C} \cdot \overline{A\bar{B}\bar{C}D} = A\bar{C}(B + \bar{D})$.

The missing square can be considered in terms of the remaining variables only, that is the " $\bar{B}D$ square." The group can thus be described as " $A\bar{C}$ but not $\bar{B}D$," which is written algebraically as

$$A\bar{C} \cdot \overline{\bar{B}D} = A\bar{C}(B + \bar{D})$$

In this map, there is also "almost" a group C ; there are two squares missing from this group, these squares being described as $A\bar{D}$. There is thus the group " C but not $A\bar{D}$," which written algebraically is

$$C \cdot \overline{A\bar{D}} = C(\bar{A} + D)$$

The final factored expression is

$$A\bar{C}(B + \bar{D}) + C(\bar{A} + D)$$

PROBLEMS

Minimize the following expressions using the map method.

1. $\bar{A}\bar{B}C + AD + \bar{D}(B + C) + A\bar{C} + \bar{A}\bar{D}$
- *2. $ACD + B\bar{D} + \bar{B}\bar{C} + \bar{B}D + \bar{C}\bar{D}$
Optional combinations: $\bar{A}\bar{B}C\bar{D}$, $A\bar{B}C\bar{D}$, $\bar{A}B\bar{C}D$
3. $B\bar{C} + \bar{A}B + BC\bar{D} + \bar{A}\bar{B}D + A\bar{B}\bar{C}D$
4. $B\bar{C}\bar{D} + \bar{A}\bar{B}D + A\bar{C}D + \bar{A}BC + \bar{A}B\bar{C}D$
Optional combinations: $\bar{A}\bar{B}C\bar{D}$, $ABCD$, $A\bar{B}C\bar{D}$
- *5. $C(B\bar{D} + \bar{B}D) + \bar{A}C(B + D) + CD(A + B) + AD(\bar{B} + \bar{C}) + AB$
6. Read the map in Fig. 7-32.

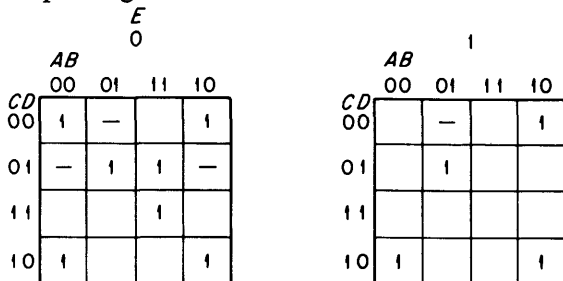


Figure 7-32

7. Read the map in Fig. 7-32, using the complementary approach.

8. Using the map method, obtain the minimum sum of products:

$$\begin{aligned} \bar{A}\bar{B}\bar{C}D\bar{E}\bar{F} + \bar{A}BCD\bar{E}\bar{F} + A\bar{B}\bar{D} + BC\bar{D}\bar{E}F + B\bar{C}\bar{D}\bar{E}\bar{F} \\ + \bar{A}\bar{B}\bar{C}\bar{D}E + \bar{A}\bar{B}\bar{C}\bar{D}F + \bar{A}CDE\bar{F} + \bar{A}\bar{C}\bar{D}\bar{E}F \end{aligned}$$

Optional combinations: $\bar{A}\bar{B}\bar{C}D\bar{E}\bar{F}$, $\bar{A}\bar{B}\bar{C}D\bar{E}\bar{F}$

*9. Using the map method, obtain the minimum sum of products:

$$\begin{aligned} AB\bar{C}D\bar{E}\bar{F} + AB\bar{D}\bar{E}F + A\bar{B}\bar{C}\bar{D}F + A\bar{C}DE\bar{F} + \bar{A}B\bar{D}\bar{E}\bar{F} \\ + \bar{A}\bar{B}\bar{C}D\bar{E}\bar{F} + \bar{A}\bar{B}\bar{C}\bar{D}E + \bar{A}\bar{C}\bar{D}\bar{E}F + \bar{B}C\bar{D} \end{aligned}$$

Optional combinations: $A\bar{B}\bar{C}D\bar{E}\bar{F}$, $\bar{A}\bar{B}\bar{C}D\bar{E}\bar{F}$

8

Trees—Relay and Electronic

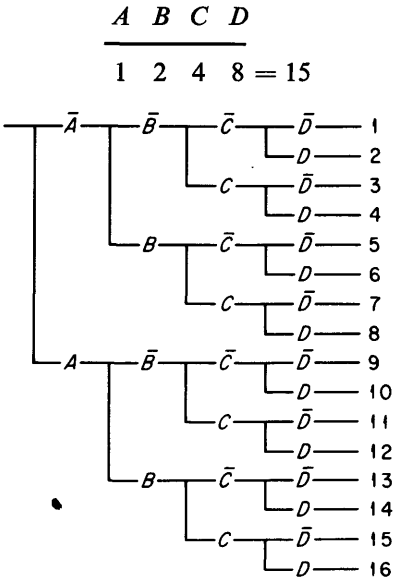
A *tree* is a multi-output circuit in which each input combination has a unique output associated with it. Two types of trees will be discussed: relay trees and electronic trees.

Relay Trees

A relay tree, or transfer tree, is a particular type of multi-terminal relay contact network having a single input which may be connected to any one of a number of outputs.¹ Only one output is connected to the input at any given time, the selection being controlled by the combination of relays operated. Each input-to-output path passes through one contact on each relay, and all outputs are disjunctive, that is, no output can ever be connected to another output through the circuit.

¹Trees are also sometimes used in reverse, that is, one of a number of inputs is connected to a single output.

The number of possible relay combinations with n relays is 2^n ; therefore, in an n -relay tree there are 2^n possible outputs. A full tree has an output terminal for each of the 2^n possible relay combinations, and the total number of transfers² in the tree is $2^n - 1$. A partial tree has less than 2^n output terminals, and the total number of transfers in the tree may vary. A full transfer tree is shown in Fig. 8-1. The transfer contact distribution on the tree in Fig. 8-1 is



By the rearrangement of a full tree, the contact load may be somewhat equalized. The relay connected to the input of the tree must always have a single transfer contact; however, the contact loads on the other relays may be made more uniform. In the above tree, there is a total of fourteen transfer contacts on relays B , C , and D . The most even contact division among these three relays is a 4—5—5 distribution.

A circuit with the transfer contact distribution

A	B	C	D
1	4	5	5

is shown in Fig. 8-2. The outputs are numbered to correspond with those in the previous tree.

Regardless of the contact distribution, the total number of transfers in

²In this chapter “transfers” denote “positions.”

a full tree is always $2^n - 1$. Rearrangement of a partial tree, however, can lead to a reduction in the total number of transfers required.

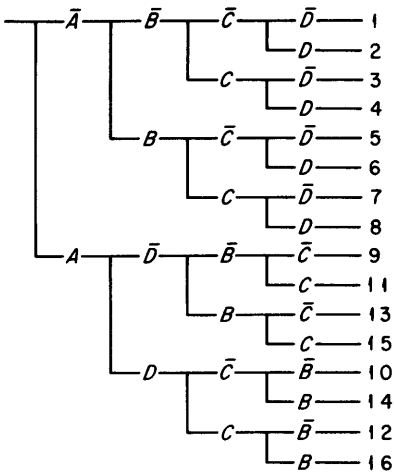


Figure 8-2

Minimization of Partial Trees

A method of minimizing partial trees, that is, obtaining a required tree with the minimum number of transfers, will now be examined. All possible arrangements of a tree could, of course, be tried, and the minimal one selected, but this process would be too long and laborious. The following table gives some indication of the progressive complexity of a trial and error approach as n increases.

Number of relays in tree	Number of possible arrangements of full tree	
2	$2^1 =$	2
3	$3^1 \cdot 2^2 =$	12
4	$4^1 \cdot 3^2 \cdot 2^4 =$	576
5	$5^1 \cdot 4^2 \cdot 3^4 \cdot 2^8 =$	1,658,880
6	$6^1 \cdot 5^2 \cdot 4^4 \cdot 3^8 \cdot 2^{16} =$	16, 511, 297, 126, 400

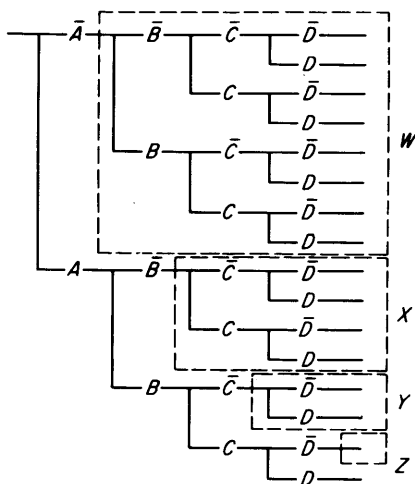
The number of possible arrangements of an n -relay tree is represented by

$$[n^{2^0}][(n-1)^{2^1}][(n-2)^{2^2}][(n-3)^{2^3}] \dots [2^{2^{n-1}}]$$

If P represents the number of possible arrangements of an n -relay tree,

then the number of possible arrangements of an $(n + 1)$ relay tree is $(n + 1)P^2$.

In a full transfer tree, such as the one shown in Fig. 8-3, a branch, or minor tree, representing 2^x combinations contains $2^x - 1$ transfers.



Branch *W* represents 8 combinations and contains 7 transfers
 Branch *X* represents 4 combinations and contains 3 transfers
 Branch *Y* represents 2 combinations and contains 1 transfer
 Branch *Z* represents 1 combination and contains 0 transfers

Figure 8-3

A particular partial tree can be obtained by starting with a full tree and removing the branches representing the groups of unused output combinations. The removal of a branch representing a group of 2^x unused combinations results in the elimination of $2^x - 1$ transfers. For instance, in the tree in Fig. 8-3, if the four combinations $A\bar{B}\bar{C}\bar{D}$, $A\bar{B}\bar{C}D$, $A\bar{B}C\bar{D}$, and $A\bar{B}CD$ were unused, the branch labeled (*X*), representing the group of these four combinations, could be removed, eliminating three transfers.

It follows that the total number of transfers eliminated from a full tree equals the number of unused combinations minus the number of groups into which these combinations are combined. The method of minimizing a partial tree, therefore, consists of starting with a hypothetical full tree and eliminating the maximum number of transfers by arranging the order of the relays in the tree so that the unused combinations can be combined into the minimum number of groups. The key to the method, then, lies not in the analysis of the used relay combinations, but rather in the analysis of the unused combinations.

Maps are used as the means for combining the unused combinations and obtaining the optimum order of the relays in the tree. The following differences between the normal use of maps and their use here should be noted: in this method (1) the unused rather than the used combinations are of prime consideration, (2) each combination is not considered as often as desired but is considered only once, and (3) the groups formed must be compatible with fundamental transfer tree configuration. The meaning of this third point will be apparent presently.

The use of the map to obtain any desired partial n -relay tree (not necessarily a minimal one) will be described first. An n -variable map is drawn, and a 1 is entered in each square representing an output combination, as shown in Fig. 8-4.

Output Combinations

$\bar{A}\bar{B}\bar{C}\bar{D}$
 $\bar{A}\bar{B}\bar{C}D$
 $\bar{A}\bar{B}C\bar{D}$
 $\bar{A}\bar{B}CD$
 $\bar{A}B\bar{C}\bar{D}$
 $\bar{A}B\bar{C}D$
 $\bar{A}BC\bar{D}$
 $\bar{A}BCD$

	AB			
	00	01	11	10
CD				
00	1	1	1	
01	1	1		
11				1
10			1	1

Figure 8-4

The map is then divided into two $(n - 1)$ -variable submaps, the divided variable becoming an adjunct to one submap, and the complement of the divided variable becoming an adjunct to the other submap. In each submap, the adjunct to that submap is written in the lower left-hand corner of each square representing an output combination (Fig. 8-5). (The particular order of subdivision used in this example leads to a minimal tree; the basis for arriving at this optimum order will be apparent later, and the example should be reviewed from this standpoint.)

	AB			
	00	01	11	10
CD				
00	1 \bar{A}	1 \bar{A}	1 A	
01	1 \bar{A}	1 \bar{A}		
11				1 A
10			1 A	1 A

Figure 8-5

	AB			
	00	01	11	10
CD				
00	1 $\bar{A}\bar{C}$	1 $\bar{A}\bar{C}$	1 AB	
01	1 $\bar{A}\bar{C}$	1 $\bar{A}\bar{C}$		
11				1 AB
10			1 AB	1 AB

Figure 8-6

Each $(n - 1)$ -variable submap containing an output combination is subdivided into two $(n - 2)$ -variable submaps, the newly-formed adjunct in each case being written to the right of the previously-written adjunct (Fig. 8-6).

A submap containing only unused combinations is not further subdivided. An unused-combination submap containing 2^x squares represents the elimination of $2^x - 1$ transfers.

The subdivision process is continued until each 1-square becomes a submap; all other submaps will contain only unused combinations (Fig. 8-7).

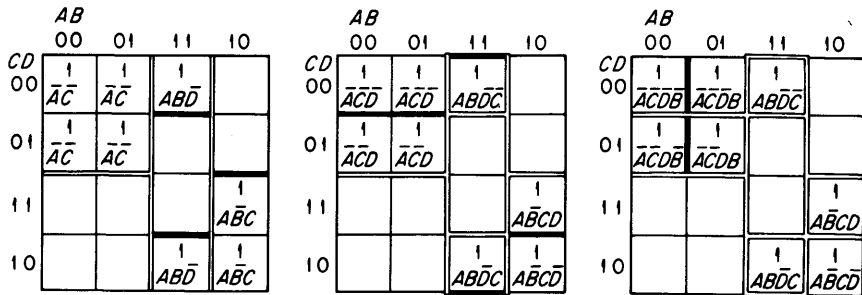


Figure 8-7

The designation of each output combination will be completely written at the bottom of its representative square. The relay tree can then be drawn with reference to these written designations, each input-to-output path from left to right corresponding to the equivalent order of the related written combination (Fig. 8-8). Note that in a partial tree, sometimes only the normally-open or only the normally-closed part of a transfer contact is used.

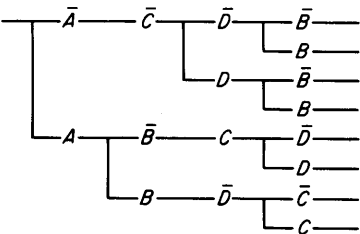


Figure 8-8

The order of subdivision determines the order of the relay contacts in the tree. The writing of the adjuncts in the 1-squares is the means of recording the order of subdivision so that the transfer tree can be drawn by reference to the map. The subdivision process insures that the submaps formed will be compatible with transfer tree configuration.

Any order of subdivision will lead to a legitimate tree. However, to obtain a desired tree having the minimum number of transfers, the map must be subdivided so that the number of unused-combination submaps is a minimum. The procedure is to combine the unused combinations into the minimum number of groups that can be obtained by the subdivision

process³ (there may be more than one way of doing this), and then subdivide the map to form the desired groups (there may be more than one way of doing this, also).

This was the procedure followed in the example, as a review will illustrate. Analysis of the map shows that the minimum possible number of groups of unused combinations is three ($\bar{A}\bar{C}$, $AB\bar{D}$, and $A\bar{B}\bar{C}$), and that these groups can be obtained by the subdivision process. To form these three groups, the map was subdivided as shown. The tree obtained is, therefore, a minimal one.

The method may be modified by continuing the subdivision process only until all unused-combination submaps have been formed, which completes the branch removal (Fig. 8-9).

From the map at this stage, the tree can be partially constructed, and the rest completed arbitrarily. The portion of the tree obtained from this map is shown in Fig. 8-10. Branch removal has been completed and any variations in the rest of the tree will not affect the number of transfers.

	AB			
	00	01	11	10
AD				
00	$\bar{A}\bar{C}$	$\bar{A}\bar{C}$	$AB\bar{D}$	
01	$\bar{A}\bar{C}$	$\bar{A}\bar{C}$		
11				$A\bar{B}\bar{C}$
10			$AB\bar{D}$	$A\bar{B}\bar{C}$

Figure 8-9

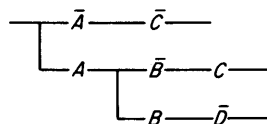


Figure 8-10

Following is a summary of the relationships involved:

n = number of relays in tree.

2^n = number of output combinations in full tree.

$(2^n - 1)$ = number of transfers in full tree.

m = number of unused output combinations.

³In the facing map, it is possible to combine the unused combinations into three groups ($\bar{A}\bar{C}$, $BC\bar{D}$, $A\bar{B}\bar{D}$). However, these three groups cannot be collectively obtained by the subdivision process (they are not compatible with transfer tree configuration). The minimum number of groups of unused combinations that can be obtained by the subdivision process is four.

	AB			
	00	01	11	10
CD				
00			1	1
01			1	
11	1	1	1	
10	1			1

Figure 8-11

$(2^n - m)$ = number of used output combinations.

p = number of unused combination submaps.

$(m - p)$ = number of transfers eliminated from full tree.

$(2^n - 1) - (m - p) = (2^n - m) + p - 1$ = number of transfers in partial tree.

Electronic Trees

The block diagram of Fig. 8-12 represents an electronic tree. As in relay trees, each combination of inputs turns on one of the possible outputs. A straightforward way of accomplishing this switching is shown in Fig. 8-13.

Using the total number of logic block inputs as a measure of circuit cost, the circuit "costs" sixty-four inputs (sixteen four-input AND's).

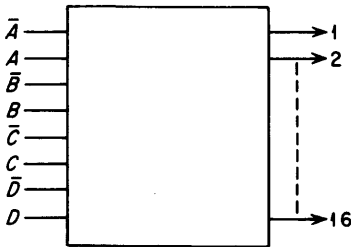
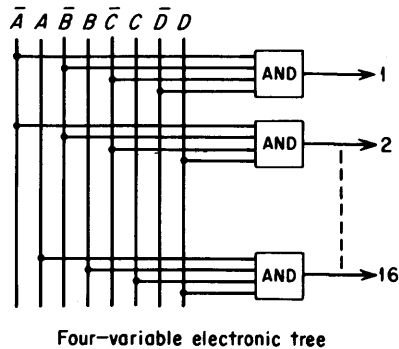


Figure 8-12



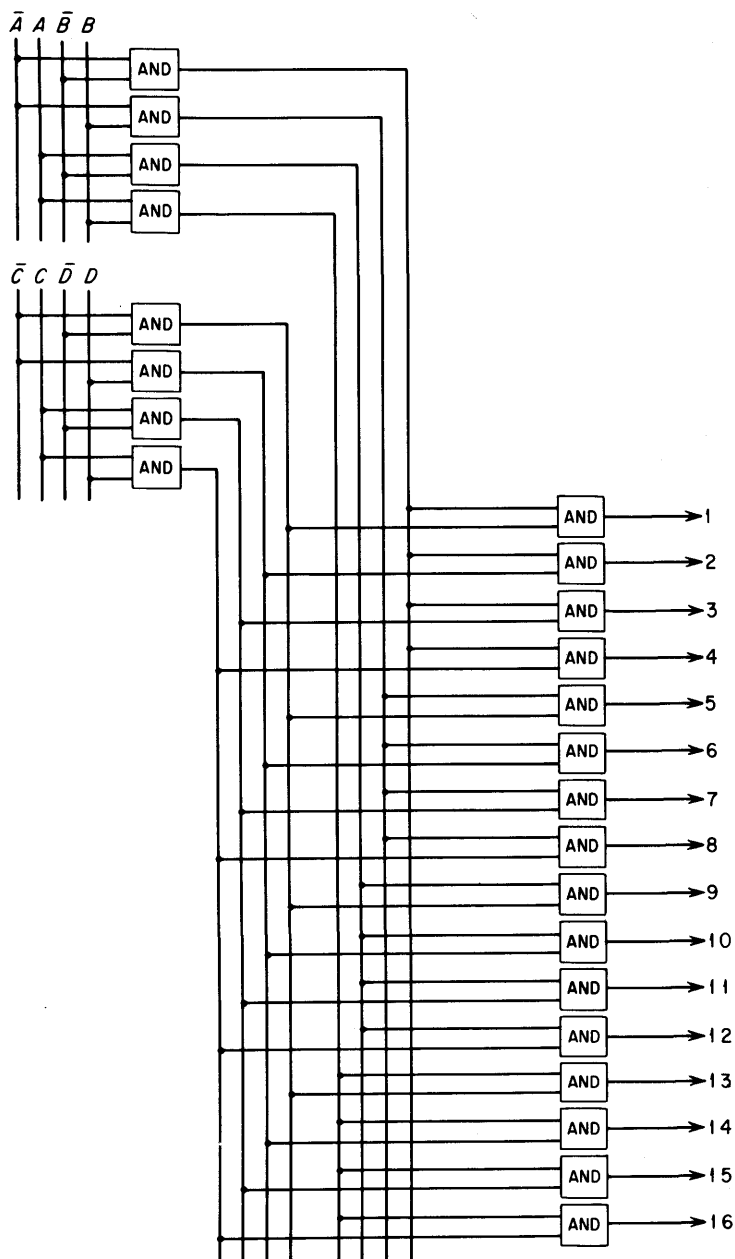
Four-variable electronic tree

Figure 8-13

The circuit in Fig. 8-14 accomplishes the same function as the previous circuit, but with only forty-eight inputs (twenty-four two-input AND's). This is the most economical circuit accomplishing the desired switching.

A method for obtaining an optimum network of this type, regardless of the number of variables involved, will now be discussed.

The total number of variables is written, and divided as evenly as possible into two numbers (integers). This is diagrammed by writing the number of variables, and from it drawing two lines to the left, each line terminating at one of the numbers into which it has been divided. The resultant numbers are divided the same way, this process being continued until the numbers 2 or 3 are reached. Two lines are drawn from each 2, and three lines from each 3, these lines terminating in the variables involved. This process is illustrated in Fig. 8-15, with an example in nine variables.



Most economical four-variable electronic tree

Figure 8-14

Such a diagram is interpreted by reading from left to right as follows: each 2 represents two of the variables that are switched in all possible (four) combinations; each 3 represents three of the variables that are switched in all possible (eight) combinations. In this example, the nine variables are broken down into four groups of 2, 2, 2, and 3 variables each.

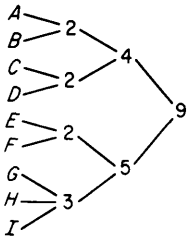


Figure 8-15

Two of the variables (*A* and *B* in this example) having been switched together in all (four) combinations, and two others (*C* and *D*) having been switched in all (four) combinations, the resultant outputs of these two circuits are switched together in all (sixteen) combinations. This operation leads to the 4 on the diagram, which represents the fact that at this point four variables (*A*, *B*, *C*, and *D*) are switched in all (sixteen) combinations. Two other variables (*E* and *F*) are switched in all (four) combinations, and the remaining three variables (*G*, *H*, and *I*) are switched in all (eight) combinations, and the resultant outputs of these two circuits are switched together in all (thirty-two) combinations. The 5 on the diagram represents the fact that five variables (*E*, *F*, *G*, *H*, and *I*) are switched in all (thirty-two) combinations at this point. Finally, the (sixteen) outputs of the four variables (*A*, *B*, *C*, and *D*) and the (thirty-two) outputs of the five variables (*E*, *F*, *G*, *H*, and *I*) are switched together in all (512) combinations, giving the 512 possible outputs with nine variables.

The total number of logic block inputs can be conveniently counted directly from the diagram. Each number, *X*, on the diagram represents the number of variables being switched in all combinations at that point. The total number of combinations at that point is therefore 2^x . In other words, 2^x is the number of AND circuits associated with the number *X* on the diagram. The number of lines leading to *X* represents the number of inputs to each AND circuit at that point. Therefore, the total number of logic block inputs associated with each number *X* equals the number of lines leading to *X* times 2^x .

Only in the case of 3's is 2^x multiplied by three; in all other cases 2^x is multiplied by two. In this example, the total number of logic block inputs is

$$\begin{array}{rcl} 2 \times 2^2 & = & 8 \\ 2 \times 2^2 & = & 8 \\ 2 \times 2^2 & = & 8 \\ 3 \times 2^3 & = & 24 \\ 2 \times 2^4 & = & 32 \\ 2 \times 2^5 & = & 64 \\ 2 \times 2^9 & = & 1024 \\ \hline & & 1168 \end{array}$$

The number of logic block inputs required in the straightforward approach is $n2^n$, where n is the number of variables. With nine variables,

$$n2^n = 9 \cdot 2^9 = 4608$$

The reader may find it interesting to compare, for other values of n , the number of logic block inputs required with each approach.

One more example, with $n = 7$, is given for study in Fig. 8-16.

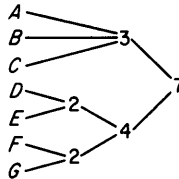


Figure 8-16

The number of logic block inputs is

$$\begin{array}{rcl} 3 \times 2^3 & = & 24 \\ 2 \times 2^2 & = & 8 \\ 2 \times 2^2 & = & 8 \\ 2 \times 2^4 & = & 32 \\ 2 \times 2^7 & = & 256 \\ \hline & & 328 \end{array}$$

With the straightforward approach, the number of logic block inputs is

$$7 \cdot 2^7 = 896$$

If all possible outputs are used, the “cost” of the most economical network is, of course, the same regardless of how the variables are grouped. However, if all of the outputs are not used, the choice of variables in each group can affect the circuit cost, as illustrated in Fig. 8-17.

In this example, a saving of three two-input AND’s results if the variables are grouped AD and BC , rather than AB and CD . The reader may wish to investigate the result of grouping AC and BD .

PROBLEMS

1. Equalize, as much as possible, the contact load on a full tree of five relays A , B , C , D , and E .

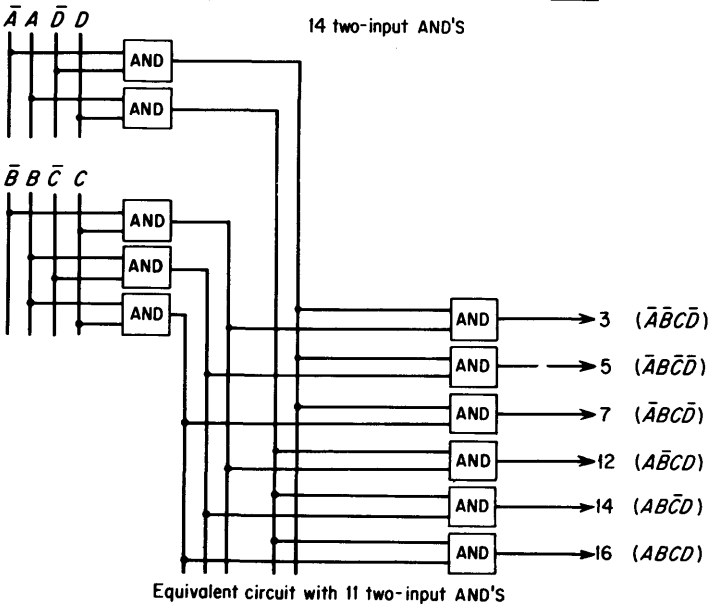
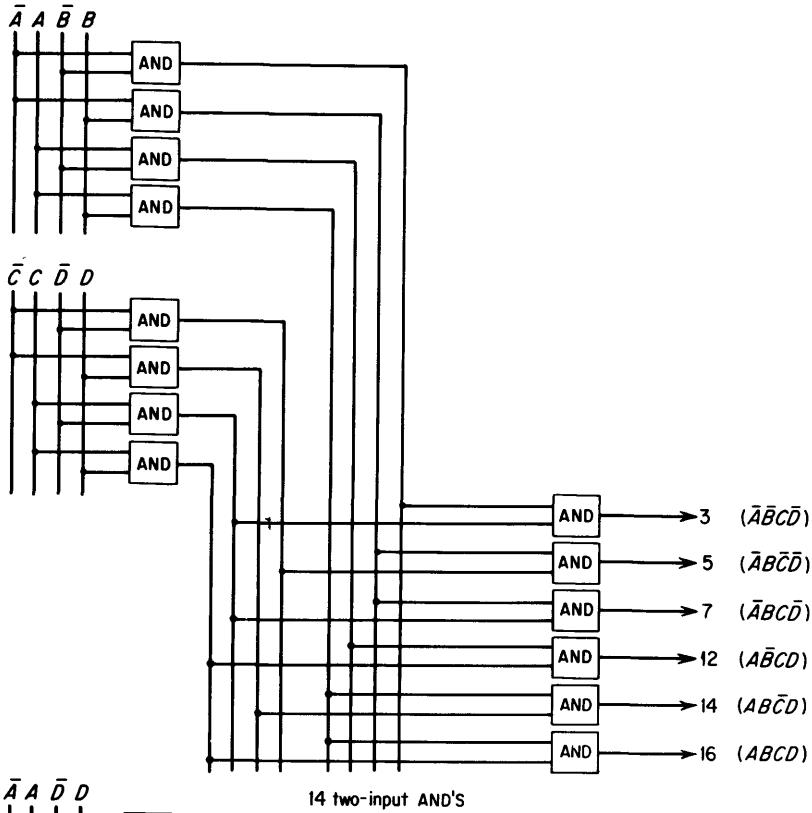


Figure 8-17

2. Minimize the number of transfers in the following partial relay trees.

(a) Output Combinations

ABCD

0000
0010
0011
0101
0111
1000
1110
1111

(b) Output Combinations

ABCD

0100
0110
1001
1011
1100
1101
1110
1111

3. The map in Fig. 8-18 shows the output combinations of a partial relay tree. How many transfers are required if

- (a) relay *A* is connected to the input?
(b) relay *B* is connected to the input?
(c) relay *C* is connected to the input?
(d) relay *D* is connected to the input?

	<i>AB</i>			
	00	01	11	10
<i>CD</i>				
00			1	1
01		1	1	
11		1	1	
10		1	1	

Figure 8-18

- *4. The map in Fig. 8-19 shows the output combinations of a partial relay tree. How many transfers are required if

- (a) relay *A* is connected to the input?
- (b) relay *B* is connected to the input?
- (c) relay *C* is connected to the input?
- (d) relay *D* is connected to the input?

	<i>AB</i>			
	00	01	11	10
<i>CD</i>				
00		1	1	
01		1	1	1
11	1			1
10		1	1	

Figure 8-19

- 5. Design the most economical electronic trees for five, six, eight, and ten variables. How many logic block inputs are saved over the straight-forward approach in each case?
- *6. Design the most economical electronic trees for eleven and thirteen variables. How many logic block inputs are saved over the straight-forward approach in each case?

9

Symmetric Functions

Design a circuit that will be closed if and only if exactly three out of a total of eight relays are operated.

An understanding of symmetric functions is useful in the design of switching circuits, particularly relay contact networks (see example above), where symmetric switching functions lead directly to bridge and non-planar networks that are much more economical than the best series-parallel circuit otherwise obtainable.

Variables of Symmetry

The function

$$XY\bar{Z} + X\bar{Y}Z + \bar{X}YZ$$

is said to be symmetric in X , Y , and Z since the successive interchanges of

any two of the variables X , Y , and Z leaves the function unaltered. The interchanging of X and Z , for example, that is, the replacement of all X 's with Z 's, all \bar{X} 's with \bar{Z} 's, all Z 's with X 's, and all \bar{Z} 's with \bar{X} 's, results in

$$ZY\bar{X} + Z\bar{Y}X + \bar{Z}YX$$

which is identical with the original function. X , Y , and Z in this function are called the *variables of symmetry*. A symmetric function is defined as one in which the interchange of any of the variables of symmetry leaves the function identically the same.

In the preceding function, all variables of symmetry were uncomplemented. Sometimes, in a symmetric function, some of the variables of symmetry may be complemented. For example, the function

$$X\bar{Y}Z + \bar{X}YZ + \bar{X}\bar{Y}\bar{Z}$$

is symmetric in X , Y , and \bar{Z} , that is, X , Y , and \bar{Z} are the variables of symmetry. Again, the interchanging of any two of the variables of symmetry will result in the identical function. For instance, interchanging X and \bar{Z} (replacing all X 's with \bar{Z} 's, all \bar{X} 's with Z 's, all Z 's with \bar{X} 's, and all \bar{Z} 's with X 's) results in

$$\bar{Z}\bar{Y}\bar{X} + ZY\bar{X} + Z\bar{Y}X$$

which is identical to the original expression.

The recognition of symmetric functions in which some of the variables of symmetry are complemented is usually not obvious, and this subject will be taken up later in this chapter.

m-out-of-n Functions

Symmetric functions in which all of the variables of symmetry are uncomplemented are usually known as such in advance by the circuit specifications and are called *m-out-of-n* functions. Algebraically, these functions equal 1 if exactly m out of the n variables equal 1.

For example, the function

$$\bar{A}BC + A\bar{B}C + AB\bar{C}$$

can be described as a "symmetric 2-out-of-3 function of the variables, A , B , and C ," and can be written in "symmetric notation" as

$$S_2^3 ABC$$

A , B , and C are the variables of symmetry, and the expression will equal 1 when exactly two of the three variables equal 1 and under no other conditions.

Symmetric functions may be defined by multiple m 's. For example, the function

$$XY + XZ + YZ$$

equals 1 only if two or three of the variables equal 1. This function can be written in symmetric notation as

$$S_{2,3}^3 XYZ$$

and can be described as a symmetric 2- or 3-out-of-3 function of the variables X , Y , and Z .

Boolean Operations with Symmetric Notations

Boolean operations can be performed with symmetric notations; that is, expressions with the same variables of symmetry can be ANDed and ORed, and these expressions or the variables of symmetry or both can be complemented. First, the ANDing of symmetric functions will be examined.

EXAMPLE:

$$(S_{1,2,4}^5 ABCDE)(S_{2,3,4}^5 ABCDE) = S_{2,4}^5 ABCDE$$

$S_{1,2,4}^5 ABCDE$ equals 1 if one, two, or four of the variables of symmetry equal 1. $S_{2,3,4}^5 ABCDE$ equals 1 if two, three, or four of the variables of symmetry equal 1. For the product to equal 1, both terms must equal 1, and this can occur only if either two or four of the variables of symmetry equal 1.

Thus, ANDing two symmetric functions containing the same variables of symmetry is accomplished by retaining those subscripts common to both terms. If there are no subscripts in common, the product, of course, equals 0.

Next, the ORing of symmetric functions will be examined.

EXAMPLE:

$$S_{1,2,4}^5 ABCDE + S_{2,3,4}^5 ABCDE = S_{1,2,3,4}^5 ABCDE$$

The sum of the two terms will equal 1 if either term equals 1, that is, if one, two, three, or four of the variables of symmetry equal 1. Thus, in ORing two symmetric functions, all of the subscripts in both terms appear in the final expression. If every possible subscript, from 0 through n , occurs, the sum equals 1.

Now, the complementation of a symmetric function will be discussed.

EXAMPLE:

$$\overline{S_{1,2,3,4}^5 ABCDE} = S_{0,5}^5 ABCDE$$

The function $\overline{S_{1,2,3,4}^5 ABCDE}$ is *not* equal to 1 (is equal to 0) if one, two,

three, or four of the variables of symmetry equal 1. It is logically equivalent to say that this function equals 1 for any condition other than one, two, three, or four of the variables of symmetry equalling 1. The only other possible conditions are none or five of the variables of symmetry equalling 1.

Thus, complementing a symmetric function is accomplished by supplying all subscripts, from 0 through n , that are missing from the original expression. In the example, the missing subscripts are 0 and 5.

Finally, the operation of *complementing the variables of symmetry* will be examined.

EXAMPLE:

$$S_{1,3}^5 ABCDE = S_{2,4}^5 \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$$

The expression $S_{1,3}^5 ABCDE$ equals 1 if one or three of the variables of symmetry equals 1. Saying that one of the variables of symmetry equals 1 is the same as saying that n minus one (or, in this example, four) of the variables of symmetry equal 0. Saying that three of the variables of symmetry equal 1 is the same as saying that two of the variables of symmetry equal 0.

Therefore, another way of saying that a symmetric function equals 1 if one or three of the five variables of symmetry equal 1 is to say that the function equals 1 if two or four of the five variables of symmetry equal 0. Still another way of saying the same thing is that the function equals 1 if two or four of the *complemented variables of symmetry* (\bar{A} , \bar{B} , \bar{C} , \bar{D} , and \bar{E}) equal 1.

Thus, another way of writing a symmetric function is to complement all of the variables of symmetry and obtain a new set of subscripts by subtracting each of the original subscripts from the total number of variables.

For practice, the equivalence of the following four symmetric expressions should be verified.

$$\begin{array}{ll} S_{1,3,4,5}^5 ABCDE & \overline{S_{0,2}^5 ABCDE} \\ S_{0,1,2,4}^5 \bar{A}\bar{B}\bar{C}\bar{D}\bar{E} & \overline{S_{3,5}^5 \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}} \end{array}$$

Symmetric Relay Contact Networks

Suppose a relay network is desired that is closed only when exactly three out of a total eight relays are operated. The Boolean expression for this circuit might start out like:

$$ABC\bar{D}\bar{E}\bar{F}\bar{G}\bar{H} + AB\bar{C}\bar{D}\bar{E}\bar{F}\bar{G}\bar{H} + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}\bar{G}\bar{H} + \dots \text{etc.}$$

for $56 ({}_8C_3)$ terms. Examination of this expression will show that no sim-

plication is possible other than factoring. However, a 3-out-of-8 circuit is a symmetric circuit, and circuits of this type can be designed in a matter of seconds, even though they may be complex bridge networks or even nonplanar networks.

First, the general structure of symmetric networks will be examined.

Symmetric Trees

A symmetric tree is a multi-output relay circuit with one input and $n + 1$ outputs, where n is the total number of relays in the circuit. The outputs are numbered from 0 through n , and with m out of n relays operated, the m output is connected to the input.

As an example, a three-relay symmetric tree is shown in Fig. 9-1, both in conventional symbolic form, and also in a diagrammatic form that is convenient to use for symmetric circuits.

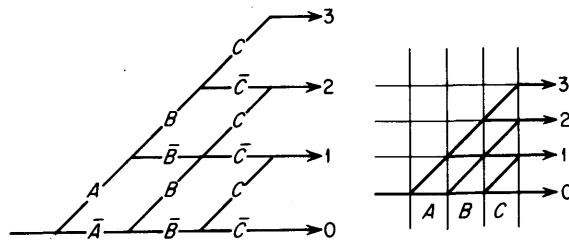


Figure 9-1

Referring first to the circuit diagram on the left, it can be seen that with zero relays operated, the input is connected to the 0 output; with one relay operated, the input is connected to the 1 output; etc. It should be obvious that a circuit structure of this type can be extended to include any number of relays. The diagram on the right represents the identical circuit. To construct such a diagram, $n + 1$ vertical guide lines and $n + 1$ horizontal guide lines are drawn, as shown. Then n relay designations are written at the bottom, in the spaces between the vertical guide lines. At the right, the horizontal guide lines are labeled, from bottom to top, with the output designations 0 to n . Horizontal and diagonal lines are then drawn as shown; all horizontal lines between two vertical guide lines represent normally-closed contacts on the relay indicated below, and all diagonal lines between the two vertical guide lines represent normally-open contacts on that relay.

For the usual symmetric circuit requirement, only a portion of a symmetric tree is required. Suppose, for instance, a circuit is desired that is closed only when exactly three out of eight relays, A through H , are

operated. This circuit requirement can be written in symmetric notation as

$$S_3^8 ABCDEFGH$$

The circuit can immediately be drawn, as shown in Fig. 9-2. Nine vertical guide lines are drawn—one more than the total number of relays involved—leaving a vertical space for each of the eight relays. Four horizontal guide lines are drawn—one more than the number of relays that must be operated for an output—the topmost guide line representing the 3 (relays operated) output. Only that portion of the symmetric tree leading to the 3 output is then drawn.

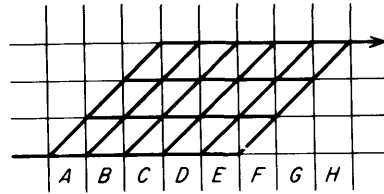


Figure 9-2

The order of the relays is arbitrary; no matter what the order, the circuit diagram remains the same. Also, the contact distribution cannot be equalized; a relay closer to either end of the diagram usually requires fewer contacts than one nearer the middle of the diagram.

Identification of Transfer Contacts

Figure 9-3 shows a method for identifying transfer contacts on symmetric circuit diagrams. A small arc is drawn between a normally-open and normally-closed contact, signifying a transfer contact. Six transfer contacts are required for the symmetric tree in Fig. 9-3.

The 3-out-of-8 circuit is redrawn in Fig. 9-4, with the transfer contacts identified. Note that on relays *A* and *H* there is one transfer contact each; on relays *B* and *G*, two transfers each; on relays *C* and *F*, three transfers each; and on relays *D* and *E*, three transfers plus one normally-closed contact each.

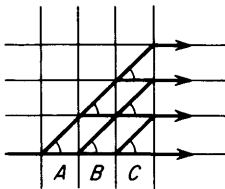


Figure 9-3

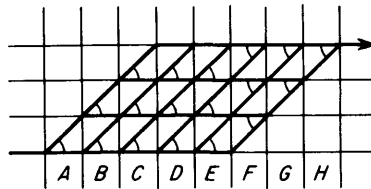


Figure 9-4

Symmetric Circuits with Multiple *m*'s

Symmetric circuits in which an output is desired for two or more different numbers of relays operated—circuits defined by symmetric notations with multiple subscripts—will now be discussed.

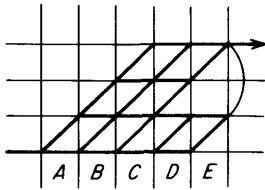


Figure 9-5

Suppose, for instance, that a circuit requirement is $S_{1,3}^5 ABCDE$. That part of a symmetric tree giving outputs for “one-out-of-five” and “three-out-of-five” can be drawn, and the outputs connected together, as shown in the diagram of Fig. 9-5.

This procedure can always be followed regardless of how many different numbers of operated relays lead to an output. However, in general, this is not the most economical way of realizing such circuit requirements, and some methods for obtaining more economical circuits will now be examined.

Elimination of redundant transfer contacts. When symmetric circuits are defined by notations with multiple subscripts forming an arithmetic progression with a difference of *one*, redundant transfer contacts can be eliminated, as illustrated by the following example.

EXAMPLE:

A $S_{2,3}^5 ABCDE$ circuit is required (Fig. 9-6).

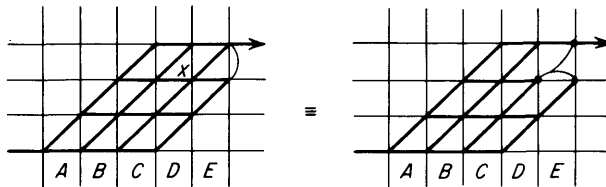


Figure 9-6

In the left-hand figure, pay particular attention to the point marked X . The circuit between X and the output is a parallel path consisting of a normally-open and a normally-closed contact on E , which is equivalent to a closed circuit. Therefore, point X can be common to the output, and these contacts on E can be eliminated, the circuit reducing to that shown on the right. All points shown in heavy dots are common output points. Note that since the remaining normally-closed and normally-open contacts on E are both common to the output, they actually form a transfer contact although not shown adjacent on the diagram.

The simplification principle illustrated in the above example can be extended to cases with more than two subscripts, as the following example shows.

EXAMPLE:

A $S_{2,3,4}^6 ABCDEF$ circuit is required (Fig. 9-7).

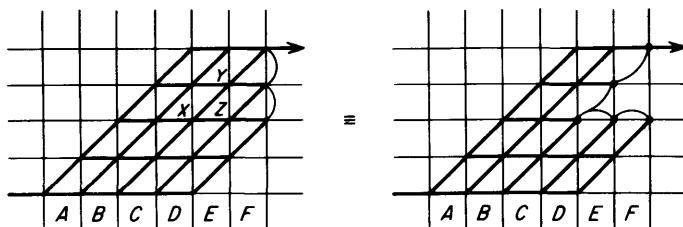


Figure 9-7

Note, in the left diagram, that there is a closed circuit between the point Y and the output, and between Z and the output; therefore, points Y and Z can be common to the output and two transfers on F can be eliminated. There is also a closed circuit between point X and the common points Y and Z . Thus, point X can also be common to the output, and a transfer on E can be eliminated. The final circuit is shown at the right.

Shifting down. If the multiple subscripts form an arithmetic progression with a difference greater than one, and the next step in the progression would be greater than the number of variables, a process called “shifting down” can be used to achieve economy.

Shifting down could not be used in the two previous examples because the arithmetic progression was not greater than one. A $S_{2,4}^0 ABCDEF$ circuit cannot be shifted down because even though the difference in the subscript progression is two, the next step in the progression would be 6, and 6 is not greater than the number of variables. Presently it will be seen why such a circuit cannot be shifted down.

A $S_{2,4}^5 ABCDE$ circuit can be shifted down, as illustrated in Fig. 9-8. First, a circuit with an output for the lowest subscript is drawn, in this case a 2-out-of-5 circuit. Then, instead of the remaining circuitry (4-out-of-5) being drawn in the usual manner, the 2-out-of-5 output is also made a 4-out-of-5 output by using normally-open contacts to shift down from the 2 level to the 1 level, as shown. Diagonal lines are still used to represent normally-open contacts; in shifting down, they slant in the opposite direction, however.

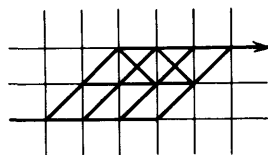


Figure 9-8

The 1 level, in a sense, now also becomes the 3 level, that is, the 1 level is connected to the input if 1 or 3 relays are operated. The 2 level then also becomes the 4 level, and the output is connected to the input if two or four relays are operated.

The following example illustrates why a shift down cannot be made if the next step in the progression is not greater than the number of variables.

EXAMPLE:

A $S_{2,4}^6 ABCDEF$ circuit is required.

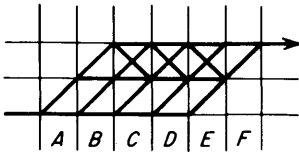


Figure 9-9

In the circuit of Fig. 9-9, a shift down has been made in an attempt to realize the required function.

An output for two or four relays operated is obtained, as desired. However, if six relays are operated, there will also be an output, which is not desired. The circuit shown in thus a $S_{2,4,6}^6 ABCDEF$ circuit.

A shift down may be made through any number of levels, as illustrated in the next example.

EXAMPLE:

A $S_{4,8}^9 ABCDEFGHI$ circuit is required.

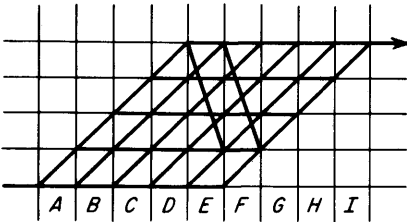


Figure 9-10

The number of levels shifted down is one less than the difference in the subscript progression. In this example, the difference is four ($8 - 4$); therefore a shift down of three levels is made (Fig. 9-10). Four relays operated connect the input to the 4 level. The normally-open contacts used for the shift down—represented by the long diagonals is columns *E* and *F*—constitute the “fifth” relay operated. Three more relays operated, totaling eight in all, again connect the input to the 4 level.

Sometimes additional levels must be added before a shift down can be made.

EXAMPLE:

A $S_{2,8}^9 ABCDEFGHI$ circuit is required.

Since the difference in the subscript progression is six, a shift down of five levels is indicated. The circuit must therefore be brought up to the 5 level before a five-level shift down can be made, as shown in Fig. 9-11.

Equivalent points. Another simplification procedure is based on the recognition of equivalent points in the circuit diagram. This procedure will be illustrated by an example.

EXAMPLE:

A $S_{1,3}^5 ABCDE$ circuit is required. Since a shift down cannot be made, the circuit in Fig. 9-12 is drawn. Note that points *W* and *Y* each have a

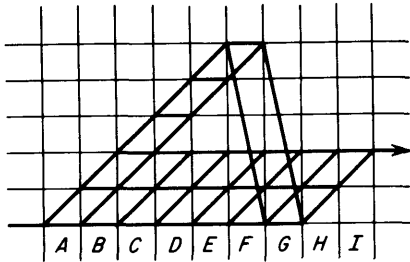


Figure 9-11

normally-closed contact on E between them and the common output. Therefore, points W and Y can be connected together, and only one normally-closed contact on E is required. Similarly, points X and Z each have a normally-open contact on E between them and the output. Therefore, points X and Z can be similarly connected together, and a normally-open contact on E eliminated (Fig. 9-13).

Carrying this same procedure further, normally-closed contacts on D connect points T and V to a common point, while normally-open contacts on D connect points T and V to another common point. Points T and V therefore can be connected and a transfer contact on D eliminated.

Normally-closed contacts on D connect points S and U to a common point; however, a normally-open contact on D connects point U to a point that is not similarly connected to point S . Therefore, points S and U are not equivalent and cannot be connected together. No other simplification is possible, and the final circuit is shown in Fig. 9-14.

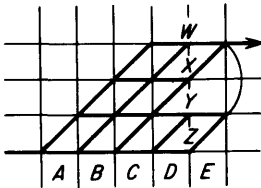


Figure 9-12

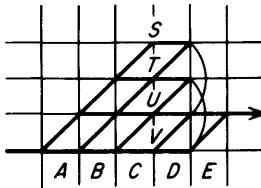


Figure 9-13

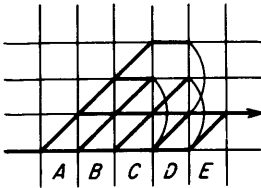


Figure 9-14

Complementation of symmetric networks. Another design approach involves graphical complementation (see Chapter 5): the complement of the desired symmetric network can be drawn, and this network graphically complemented. The network to be complemented must, of course, be planar.

EXAMPLE:

A $S_{0,1,3,4}^4 ABCD$ circuit is required.

$S_{0,1,3,4}^4 ABCD = \overline{S_2^4 ABCD}$. Therefore, a $S_2^4 ABCD$ circuit can be drawn and graphically complemented (Fig. 9-15).

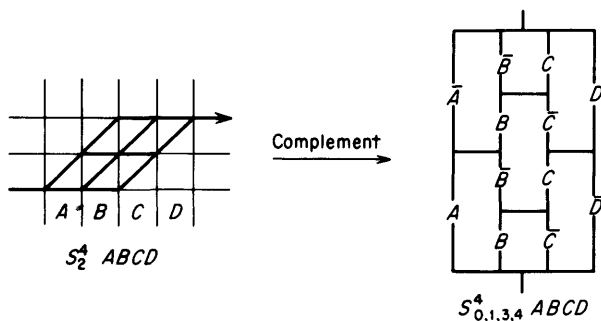


Figure 9-15

Complemented variables of symmetry. The last design approach to be considered makes use of complementing the variables of symmetry. Remember, for example, that

$$S_{1,3}^5 ABCDE = S_{2,4}^5 \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$$

In a relay network realization, the left-hand expression states that the circuit is to be closed if exactly one or three out of the five relays are operated. The right-hand expression states the same thing in a different way, namely, that the circuit is to be closed if exactly two or four out of the five relays are *unoperated*.

An advantage of complementing the variables of symmetry is that a shift down may become possible that would otherwise be impossible. For instance, a $S_{1,3}^5 ABCDE$ circuit cannot be shifted down, but the equivalent $S_{2,4}^5 \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$ circuit can be (Fig. 9-16).

In the diagram, the complemented variables of symmetry signify that the meaning of the horizontal and diagonal lines are reversed: the diagonal lines represent normally-closed contacts, and the horizontal lines represent normally-open contacts.

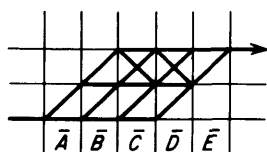


Figure 9-16

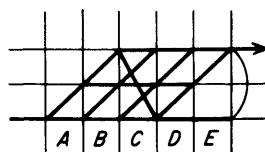


Figure 9-17

Shift downs with three or more subscripts. When, in the symmetric notation, three or more subscripts do not form an arithmetic progression, care

must be exercised in shifting down. For instance, with a $S_{0,2,5}^5 ABCDE$ circuit requirement, a shift down to make the 2 level also the 5 level is not possible since it introduces a false output for the 3-out-of-5 combination $ABC\bar{D}\bar{E}$ (Fig. 9-17). The number of variables involved may affect whether or not a shift down is possible. For example, a shift down is possible in a $S_{1,2,4}^4 ABCD$ circuit (Fig. 9-18) whereas it is not possible in a $S_{1,2,4}^5 ABCDE$ circuit because a false output for 5-out-of-5 is introduced (Fig. 9-19).

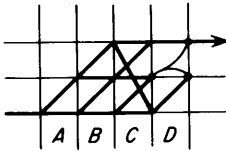


Figure 9-18

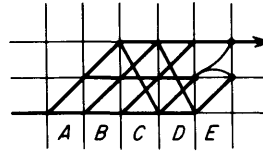


Figure 9-19

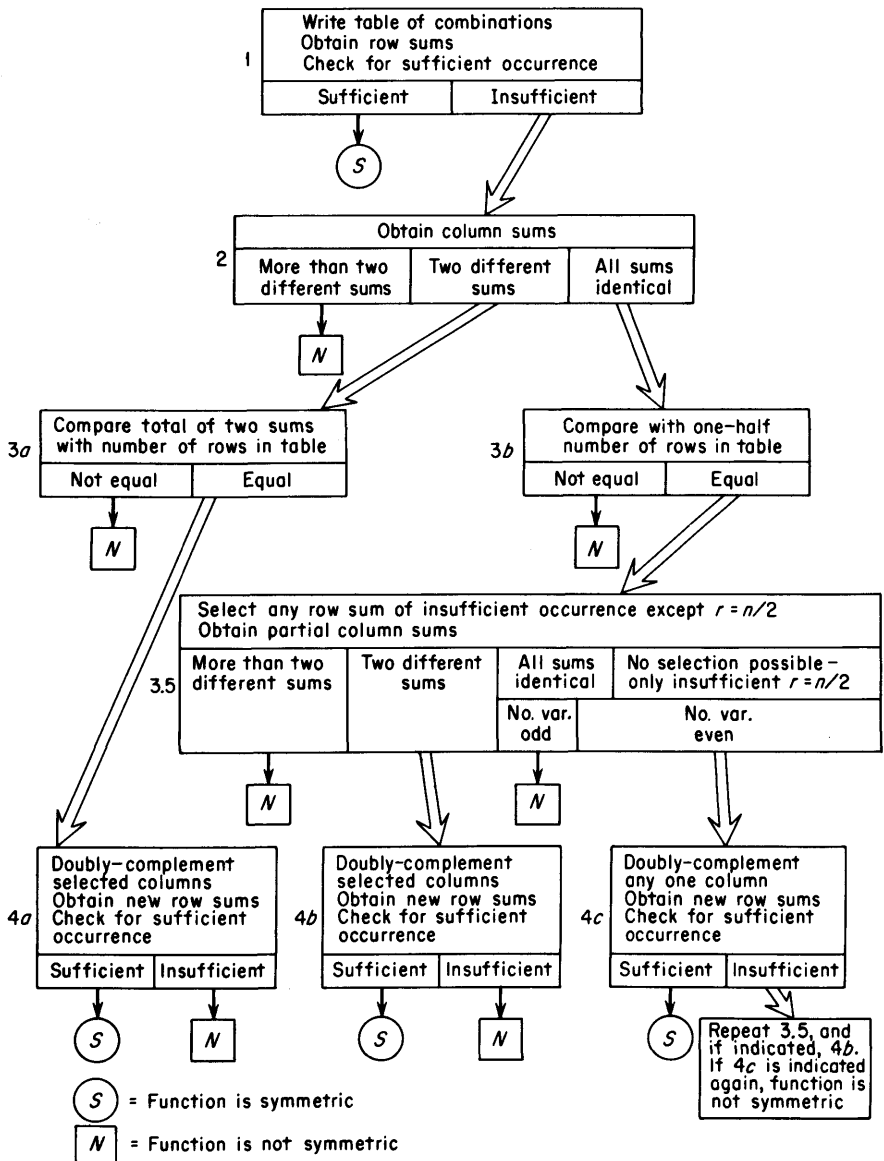
Detection and Identification of Symmetric Switching Functions

So far, the discussion of symmetric functions has been centered on the *m-out-of-n* type, in which all the variables of symmetry are uncomplemented (or all complemented). The detection and identification of symmetric functions in which any number of the variables of symmetry may be complemented will now be discussed.

In testing for symmetry, all combinations of complemented and uncomplemented variables could, of course, be investigated, but with a large number of variables such a method would be impractical. A method which uses tables of combinations for detecting and identifying symmetric switching functions will be shown. This method is given without proof, but a study of the behavior of symmetric functions manipulated in tabular form can verify the validity of the method. Figure 9-20 gives an outline of this method in diagrammatic form.

Step 1

The switching function to be tested for symmetry is written as a table of combinations, each variable appearing (uncomplemented) at the head of its respective column. The table should be checked to assure that no row combination occurs more than once. The arithmetic sum of each row in the table is obtained and written to the right of the row. All row sums are then checked for "sufficient occurrence"; if n represents the number of variables (columns) and r represents a row sum, then that row sum should occur



Outline of method for detection and identification of symmetric switching functions

Figure 9-20

$$\frac{n!}{r!(n-r)!}$$

times. This is the formula for the number of combinations of n things taken r at a time (${}_nC_r$). The following table gives the required row sum occurrences for up to eight variables. This table is an adaptation of Pascal's triangle, and may be extended to include as many variables as desired.

Table of Required Row Sum Occurrence

Row Sum	Number of Variables							
	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8
2		1	3	6	10	15	21	28
3			1	4	10	20	35	56
4				1	5	15	35	70
5					1	6	21	56
6						1	7	28
7							1	8
8								1

If all row sums occur the required number of times, the function is symmetric. The row sums represent the subscript numbers, and the variables of symmetry are denoted at the head of the columns. If all row sums do not occur the required number of times, go to Step 2.

EXAMPLE:

A	B	C	
1	1	0	2
1	1	1	3
1	0	1	2
0	1	1	2
3	3	3	

The required occurrence of row sum 2, with 3 variables, is

$$\frac{3!}{2!1!} = 3$$

The required occurrence of row sum 3, with 3 variables, is

$$\frac{3!}{3!0!} = 1$$

Both row sums 2 and 3 occur the required number of times; therefore, the function is symmetric and can be expressed as

$$S_{2,3}^3 ABC$$

Step 2

The arithmetic sum of each column in the table is obtained and written at the foot of the column. If more than two different column sums occur, the function is not symmetric. If exactly two different column sums occur, go to Step 3a. If all column sums are identical, go to Step 3b.

EXAMPLE:

A	B	C	D
0	0	1	1
0	1	1	1
1	1	1	0
1	2	3	2

This function is not symmetric because there are three different column sums.

Step 3a

The total of the two different column sums is obtained and compared with the number of rows in the table. If the total of the two sums is not equal to the number of rows in the table, the function is not symmetric. If the total equals the number of rows in the table, go to Step 4a.

Step 4a

Either of the two different column sums is selected (preferably the one of lesser occurrence) and all columns totaling the selected sum are *doubly-complemented*: all 1's in the column are changed to 0's all 0's are changed to 1's, and the variable at the head of the column is complemented. Note that the meaning of a column is not changed by double-complementation:

$$\text{if } X = 0, \text{ then } \bar{X} = 1$$

$$\text{if } X = 1, \text{ then } \bar{X} = 0$$

If the selected column sums are corrected to represent the new total for the doubly-complemented columns, all column sums will now be identical. New row sums are obtained and checked for sufficient occurrence. If all row sums occur the required number of times, the function is symmetric; otherwise, the function is not symmetric.

EXAMPLE:

<i>A</i>	<i>B</i>	<i>C̄</i>		<i>A</i>	<i>B</i>	<i>C̄</i>
1	0	1		1	0	0
0	1	1		0	1	0
0	0	0	≡	0	0	1
0	0	1		0	0	0
1	1	3		1	1	1

In this example, there are two different column sums, 1 and 3, and the total of these two sums is equal to the number of rows in the table ($1 + 3 = 4$). Column sum 3 is selected because it occurs less frequently than column sum 1, and column *C* is doubly-complemented.

Step 3b

The (identical) column sums are compared with one-half the number of rows in the table. If the sums are not equal to one-half the number of rows, the function is not symmetric. If the sums are equal to one-half the number of rows, go to Step 3.5.

Step 3.5

Any row sum of insufficient occurrence *except* $r = n/2$ is selected, and only the rows totaling this sum are considered in obtaining a *partial* arithmetic sum of each column. If more than two different partial sums occur, the function is not symmetric. If exactly two different partial column sums occur, go to Step 4b. If all partial column sums are identical, and the number of variables is odd, the function is not symmetric. If the number of variables is even, and all partial column sums are identical, or the only row sum of insufficient occurrence is equal to $n/2$ so that no row sum selection may be made, go to Step 4c.

Step 4b

Either of the two different partial column sums is selected (preferably

the one of lesser occurrence) and all columns *in the entire table* that contain this partial sum are doubly-complemented. New row sums are obtained and checked for sufficient occurrence. If all row sums occur the required number of times, the function is symmetric; otherwise, the function is not symmetric.

EXAMPLE:

A	B	C		A	B	C		A	B	\bar{C}	
1	0	1	2	1	0	1	2	1	0	0	1
0	1	1	2	0	1	1	2	0	1	0	1
0	0	0	0					0	0	1	1
1	1	1	3	1	1	2		1	1	0	2
1	0	0	1					1	0	1	2
0	1	0	1					0	1	1	2
3	3	3						3	3	3	

In this example, all column sums are identical and they are equal to one-half the number of rows in the table ($3 = \frac{1}{2} \times 6$). Either row sum 2 or row sum 1 may be selected since neither occurs the required number of times. Row sum 2 is arbitrarily chosen in this example. Only the first two rows—which total this row sum 2—are considered in obtaining a partial sum of each column, as shown in the middle table.

In the example, exactly two different partial column sums, 1 and 2, occur. Partial column sum 2 is selected because it occurs less frequently than partial column sum 1, and column C, in the original table, is doubly-complemented. New row sums are obtained and checked for sufficient occurrence. The function is found to be symmetric and can be expressed as

$$S_{1,2}^3 ABC\bar{C}$$

Step 4c

Any one column in the original table is selected and doubly-complemented, and new row sums are obtained and checked for sufficient occurrence. If all row sums occur the required number of times, the function is symmetric. If any row sum does not occur the required number of times, repeat Step 3.5 and, if indicated, Step 4b. If Step 4c is indicated again, it is not repeated and the function is not symmetric.

Once a symmetric function has been detected and identified, the corresponding relay contact network can be designed using the symmetric tree approach previously discussed. If a variable of symmetry is uncomplemented, the corresponding diagonal lines represent normally-open contacts

and the horizontal lines represent normally-closed contacts; if the variable of symmetry is complemented, the meaning of the lines is reversed, the horizontal lines representing normally-open contacts and the diagonal lines normally-closed contacts.

As an example, the relay contact network for the function $S_3^5 A \bar{B} C D \bar{E}$ is shown in Fig. 9-21.

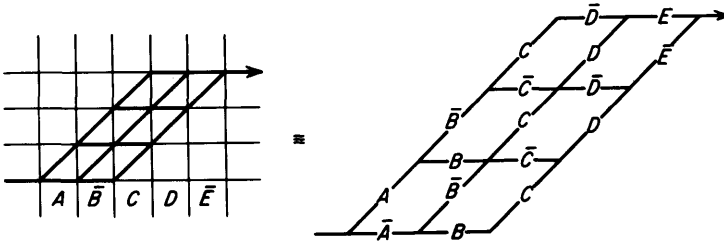


Figure 9-21

PROBLEMS

1. Design relay networks for the following symmetric functions:

- (a) $S_{3,4,5,6}^8 ABCDEFGH$
- (b) $S_{1,4}^6 ABCDEF$
- (c) $S_{1,4}^7 ABCDEFG$
- (d) $S_{1,6}^7 ABCDEFG$
- *(e) $S_{3,10}^{12} ABCDEFGHIJKL$
- *(f) $S_{1,2,4,5}^6 ABCDEF$

2. (a) $S_{1,2,4}^4 ABCD + S_{1,3,4}^4 ABCD =$

(b) $S_{0,1,3}^4 EFGH + S_{1,2,4}^4 EFGH =$

(c) $S_{1,4}^4 IJKL \cdot S_{0,3}^4 IJKL =$

(d) $\overline{S_{2,4}^4 MNOP} =$

(e) $S_{2,3}^4 QRST = S^4 \bar{Q} \bar{R} \bar{S} \bar{T}$ (Fill in missing subscripts.)

(f) $S_{0,2,4}^4 UVWX = S^4 \bar{U} \bar{V} \bar{W} \bar{X}$ (Fill in missing subscripts.)

3. Redraw the X contacts pictorially, identifying the numbered terminals.

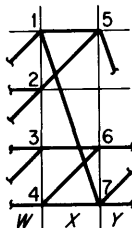


Figure 9-22

*4. Redraw pictorially.

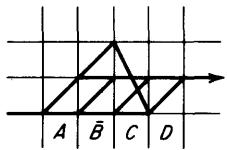


Figure 9-23

5. Detect and identify symmetry, if any:

(a) <i>ABCDE</i>	(b) <i>ABCD</i>	(c) <i>ABCD</i>	*(d) <i>ABCDEF</i>
00010	0001	0000	000110
00100	0010	0110	001001
00111	0111	0101	010010
01110	1000	1010	010100
10000	1101	1001	100001
10011	1110	1111	101000
10101			010111
11010			011110
11100			101011
11111			101101
			110110
			111001

10

Reiterative Networks

Forty relays are numbered consecutively from 1 to 40. Design a circuit that will be closed if and only if one set of three consecutive relays is operated and all other relays are unoperated.

A circuit of the type required above is called a *positional* circuit because each relay occupies a definite physical position relative to the other relays. Such a circuit is also called a *reiterative* (or iterative) circuit because of its inherent repetitive pattern. Because of this repetitive pattern, these circuits can be designed by a procedure quite different from those that have been discussed so far.

Symmetric networks fall under the class of reiterative networks also, and can be designed using the same procedure as that for positional networks. However, symmetric networks are more commonly designed using the symmetric tree approach that has been previously discussed. For this reason, the examples in this chapter will deal exclusively with the design of positional networks.

Positional requirements may specify *the number of sets* of consecutive operated or unoperated relays, without specifying the number in each set.

EXAMPLE:

A circuit of twenty relays is to be closed if and only if three sets of consecutive relays are operated and all other relays are unoperated. This circuit should be closed, for instance, if relays 2, 3, and 4; 6, 7, 8, and 9; and 15 and 16 were operated, and all others were unoperated.

A positional requirement may also specify the *number of consecutive relays* that must be operated or unoperated in each set, without specifying the number of sets.

EXAMPLE:

A circuit of twenty relays is to be closed if and only if all sets of consecutive relays operated contain three relays. This circuit should be closed, for instance, if relays 4, 5, and 6; and 13, 14, and 15 were operated, and all others were unoperated.

A positional requirement may specify both the number of sets and the number of consecutive relays operated or unoperated in each set.

EXAMPLE:

A circuit of twenty relays is to be closed if and only if there are two sets of consecutive relays operated, one set containing three relays and the other set containing two relays. This circuit should be closed, for instance, if relays 8 and 9; and 18, 19, and 20 were operated, and all others were unoperated.

Design of Reiterative Networks

In the design of reiterative networks the number of relays involved is immaterial, and the contact configuration for only one relay—which can be thought of as a prototype relay—is designed.¹ In this contact configuration, the number of input lines and the number of output lines are the same. This configuration is then repeated for all relays, and the configurations of the various relays are “strung together,” the output lines of one relay circuit becoming the input lines to the following relay circuit.

The prototype relay circuit can be thought of as being one somewhere in the middle of a chain of identical relay circuits. The prototype input lines carry pertinent information concerning all of the preceding relays. The prototype output lines, which are the input lines to the following relay, carry similar information except that now the information pertains to the prototype as well as to all of the preceding relays.

¹Sometimes economy can be achieved by using a prototype of two or more relays.

The design problem is then: (1) how many lines must carry information from relay circuit to relay circuit, and what should this information be; and (2) how should the prototype relay circuit modify this information by properly connecting the input lines to the output lines?

Sample Problem

The design procedure will be explained by working through a sample problem: A circuit of forty relays is to be closed if and only if one set of three consecutive relays is operated and all other relays are unoperated.

Thinking of the prototype relay as being somewhere in the middle of the "string" of relays, we ask, "How many different types of conditions of the relays preceding the prototype can lead to a final circuit output?"

1. If *none* of the preceding relays have been operated, it is still possible to get a circuit output.
2. If just one of the preceding relays is operated, and it is the *one immediately preceding the prototype*, a circuit output can be realized. This preceding relay would represent the first of a possible set of three consecutive relays operated.

If there is one relay operated and it is not the one immediately preceding the prototype, it must have been followed by one or more unoperated relays, creating a "set of one relay operated." Such a condition violates the circuit requirement, that is, if such a condition exists there should be no circuit output.

3. If just two preceding relays are operated it is possible to obtain a circuit output only if they are the *two consecutive relays immediately preceding the prototype*. The reasoning here is similar to condition 2, these preceding relays representing the first two of a possible set of three consecutive relays operated.

4. Only one more possible condition may lead to a circuit output: *three consecutive relays operated and all others unoperated*. In this case, the relays may or may not immediately precede the prototype.

Thus, there are four different possible conditions that can lead to a circuit output. There will be an input line to the prototype relay circuit, and a corresponding output line, for each of these conditions. Figure 10-1 represents the progress so far.

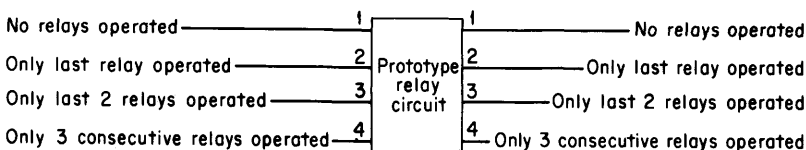


Figure 10-1

The input and output lines have been numbered for reference. Note that the input and output lines are identical, the input lines carrying information concerning all preceding relays, whereas the output lines carry information on the prototype relay as well as on all preceding relays.

The next step is to properly connect the input and output lines with contacts on the prototype relay. These connections can be indicated in a table constructed as follows: there is a row for each input condition, each row being labeled with the corresponding reference number; and there are two columns, labeled 0 and 1, representing the two possible states of the prototype relay—unoperated and operated respectively.

		0	1
No relays operated	1		
Only last relay operated	2		
Only last 2 relays operated	3		
Only 3 consecutive relays operated	4		

The entry in a given row and column will be the number of the output line that should connect to the input line for that row when the prototype relay is in the state indicated by that column.

To fill in the table, first consider input line 1—*no relays operated*. If the prototype relay is unoperated—column 0—input line 1 must be connected to output line 1—*no relays operated*; a 1 (for output line 1) is therefore entered in row 1, column 0. Output line 1, in representing *no relays operated*, includes the prototype relay as well as all preceding relays.

If the prototype relay is operated—column 1—input line 1 must be connected to output line 2—*only last relay operated*; therefore, a 2 (for output line 2) is entered in row 1, column 1. Output line 2 represents *only last relay operated* since only the prototype relay is operated under the above conditions.

Input line 2 represents *only last relay operated*, that is, the relay immediately preceding the prototype is the only one operated. If the prototype relay is unoperated, a “set of one relay operated” is created. A circuit output is not wanted for this condition, and therefore in row 2, column 0, a “—” is entered, signifying that none of the four output lines should be connected to input line 2.

If the prototype relay is operated, input line 2 must be connected to output line 3—*only last two relays operated*. Thus, a 3 is entered in row 2, column 1.

By similar analysis, a “—” is entered in row 3, column 0, and a 4 is entered in row 3, column 1.

Now consider the case where input line 4—*only three consecutive relays*

operated—is connected to the circuit input. If the prototype relay is unoperated, input line 4 must be connected to output line 4. If the prototype relay is operated, either a “set of four relays operated” would be created (if the three consecutive relays operated were those immediately preceding the prototype), or a second set would be started (if the three consecutive relays operated were not immediately preceding the prototype). In either case, no circuit output is wanted and, therefore, a “—” is entered in row 4, column 1.

Finally, information concerning the final circuit output is recorded in the table. Remembering that the input and output lines are identically designated, think now about the lines *leaving the last relay* in the string. The lines that will serve as the circuit output are noted by circling the corresponding reference numbers at the left of the table. In this example, line 4 will be the only line used as the final circuit output; therefore the 4 at the left of the table is circled. The completed table follows.

	0	1
1	1	2
2	—	3
3	—	4
④	4	—

The prototype contact network is now realized by the use of normally-closed contacts to establish the connections indicated in column 0, and by normally-open contacts establishing the connections indicated in column 1. The prototype relay network is shown in Fig. 10-2.

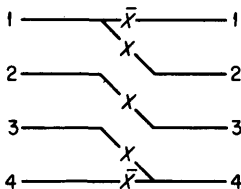


Figure 10-2

The final circuit (Fig. 10-3) is drawn by stringing together forty circuits similar to the one above. Considering any relay in the string as a prototype, if the states of all preceding relays are such that a circuit output is possible (the circuit requirements have not been violated), there will be a closed path between the circuit input and *one* of the inputs to the prototype. If the states of the preceding relays have violated the circuit requirements,

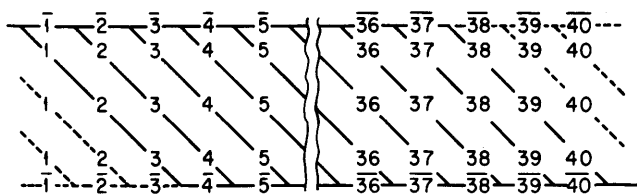


Figure 10-3

none of the prototype inputs will be connected to the circuit input.

The portions drawn lightly at both ends of Fig. 10-3 represent circuitry that is omitted. There is only one possible input to the first relay: line 1—*no relays operated*; there are only two possible inputs to the second relay: line 1—*no relays operated*, and line 2—*only last relay operated*; etc. Since line 4 is the only circuit output line, similar simplification takes place at the output end of the circuit.

Sequence Representation

To determine the number and types of input and output lines, it can be helpful to write a “string” of 0’s and 1’s representing a typical sequence leading to a circuit output, the 0’s representing unoperated relays and the 1’s representing operated relays. This sequence can then be broken up into the different kinds of prototype input conditions.

EXAMPLE:

A circuit of thirty relays is to be closed if and only if exactly two sets of consecutive relays are operated. There may be any number of relays in each set.

A typical sequence leading to a circuit output might look like

000111100000110000

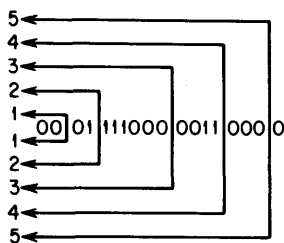


Figure 10-4

The actual number of relays in the final circuit need not be considered in writing such a typical sequence.

The different kinds of prototype input conditions can be identified by “looking back” at various points in the sequence. In effect, the prototype relay is pictured at these different points in the sequence, each breakdown including all relays preceding the prototype.

One way of breaking up the typical sequence is shown in Fig. 10-4. There are five different prototype input conditions which are identified.

The five conditions and a word statement describing each are shown below.

00	1	No relays operated
0001	2	Only one set of consecutive relays operated, including the relay immediately preceding the prototype
0001111000	3	Only one set of consecutive relays operated, not including the relay immediately preceding the prototype
00011110000011	4	Only two sets of consecutive relays operated, including the relay immediately preceding the prototype
00011110000011000	5	Only two sets of consecutive relays operated, not including the relay immediately preceding the prototype

A table is now constructed, as in the previous problem.

	0	1
1	1	2
2	3	2
3	3	4
④	5	4
⑤	5	—

Note, in this example, that there are two lines, 4 and 5, leading to the final circuit output. If the last relay is operated, as part of the second set, line 4 leads to the circuit output; if the last relay is unoperated, that is, if the second set does not include the last relay operated, line 5 leads to the circuit output. The prototype relay configuration is shown in Fig. 10-5.

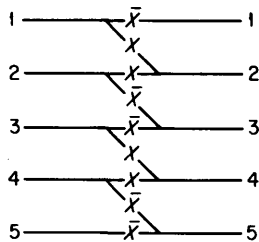


Figure 10-5

Elimination of Redundant Input Lines

Sometimes, in the design of reiterative networks, more lines than necessary are inadvertently introduced. A method of eliminating such redundancy will now be described.

Two lines are equivalent if (1) they lead to the same final circuit output

condition, that is, they both lead to a circuit output or neither leads to a circuit output, and (2) for each state of the prototype, the two lines lead to the same or equivalent lines. If two lines are equivalent, one of them is redundant and may be eliminated.

Following are shown some basic examples of equivalence. In all examples, only a portion of a table is shown, and lines 1 and 2 lead to the same final circuit output condition. In all examples, $1 \equiv 2$.

1.	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>4</td><td>3</td></tr><tr><td>2</td><td>4</td><td>3</td></tr></table>		0	1	1	4	3	2	4	3	\equiv	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>4</td><td>3</td></tr></table>		0	1	1	4	3
	0	1																
1	4	3																
2	4	3																
	0	1																
1	4	3																
2.	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>3</td></tr><tr><td>2</td><td>1</td><td>3</td></tr></table>		0	1	1	1	3	2	1	3	\equiv	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>3</td></tr></table>		0	1	1	1	3
	0	1																
1	1	3																
2	1	3																
	0	1																
1	1	3																
3.	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>2</td><td>2</td><td>3</td></tr></table>		0	1	1	2	3	2	2	3	\equiv	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>3</td></tr></table>		0	1	1	1	3
	0	1																
1	2	3																
2	2	3																
	0	1																
1	1	3																
4.	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>3</td></tr><tr><td>2</td><td>2,</td><td>3</td></tr></table>		0	1	1	1	3	2	2,	3	\equiv	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>3</td></tr></table>		0	1	1	1	3
	0	1																
1	1	3																
2	2,	3																
	0	1																
1	1	3																
5.	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>2</td><td>1</td><td>3</td></tr></table>		0	1	1	2	3	2	1	3	\equiv	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>3</td></tr></table>		0	1	1	1	3
	0	1																
1	2	3																
2	1	3																
	0	1																
1	1	3																
6.	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>—</td><td>3</td></tr><tr><td>2</td><td>—</td><td>3</td></tr></table>		0	1	1	—	3	2	—	3	\equiv	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>1</td><td>—</td><td>3</td></tr></table>		0	1	1	—	3
	0	1																
1	—	3																
2	—	3																
	0	1																
1	—	3																

It is customary to retain the smaller reference number of two equivalent lines. Therefore, in all six examples, every occurrence of a 2 is replaced by a 1. The two rows then become identical and are replaced by a single row.

Particularly note examples 4 and 5, in which "the equivalence of lines 1 and 2 is dependent upon the equivalence of lines 1 and 2." In such cases of interdependence, two lines can be made equivalent.

The subject of equivalence also enters into the design of sequential circuits, and is covered more thoroughly in Chapters 14 and 18. An example with equivalence follows.

EXAMPLE:

A circuit of ten relays is to be closed if and only if exactly one set of consecutive relays is operated, this set consisting of either one or two relays. A solution to this problem is shown in the following table.

		0	1
No relays operated	1	1	2
Only last relay operated	②	3	4
Only one relay operated, not the one immediately preceding the prototype	③	3	—
Only two consecutive relays operated	④	4	—

It is immediately seen that line 1 is not equivalent to any of the other lines since it does not lead to a circuit output while the other three lines do. Since lines 2, 3, and 4 all lead to a circuit output, these lines are examined further for equivalence.

Next, it can be seen that line 2 is not equivalent to lines 3 or 4 since, if the prototype is operated, input lines 3 and 4 do not lead to any output lines whereas input line 2 leads to output line 4.

Finally, the possible equivalence of lines 3 and 4 is examined. Both lines lead to final circuit outputs; if the prototype is operated, neither input line is connected to an output line; and if the prototype is unoperated, each input line is connected to its respective output line. Thus, lines 3 and 4 are equivalent, and every occurrence of a 4 is replaced by a 3. The third and fourth rows are now identical and are replaced by a single row. The reduced table follows.

		0	1
No relays operated	1	1	2
Only last relay operated	②	3	3
Only one relay operated, not the one immediately preceding the prototype; or only two consecutive relays operated	③	3	—

In this example, it should be intuitively seen that there is no need to differentiate between the conditions *only one relay operated, not the one immediately preceding the prototype* and *only two consecutive relays operated*, since both conditions represent “set completed,” and, in both cases, all relays beyond this point must be unoperated if a final circuit output is to be realized. The prototype relay network is shown in Fig. 10-6. Note the simplification in connecting input line 2 to output line 3, the paralleled normally-open and normally-closed *X* contacts reducing to a closed circuit.

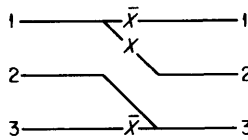


Figure 10-6

PROBLEMS

Design the prototype relay circuit for each of the following reiterative requirements: A circuit of n relays is to be closed if and only if

1. ... there is exactly one set of consecutive relays operated, this set consisting of two or more relays.
2. ... there is exactly one set of consecutive relays operated, this set consisting of one or three relays.
- *3. ... there is exactly one set of consecutive relays operated, this set consisting of two or three relays.
4. ... any set of consecutive relays operated consists of one or three relays, or if there are no relays operated.
- *5. ... any set of consecutive relays operated consists of one or three relays and there is at least one set.
6. ... there is exactly one set of consecutive relays operated, this set consisting of an odd number of relays.
7. ... there is exactly one set of consecutive relays *unoperated*, this set consisting of one or two relays.
8. ... there are exactly two sets of consecutive relays operated, both sets consisting of two relays.
- *9. ... there are exactly two sets of consecutive relays operated, one set consisting of one relay and the other set consisting of two relays. The sets may occur in either order.
10. ... there is exactly one set of three consecutive relays operated but there may be any number of sets of any other size.
11. ... there is exactly one set of consecutive relays operated, this set consisting of three relays; or there are exactly two sets, the first set consisting of one relay and the second set consisting of three relays.
12. ... there are exactly two sets of consecutive relays operated, the first set consisting of two relays and the second set consisting of three; or the first set consisting of three and the second consisting of one.

- *13.** . . . there are exactly two sets of consecutive relays operated, one set consisting of one or two relays, and the other set consisting of three relays. The sets may occur in either order.

Special Problem

A circuit of ten relays is to be *open* if and only if there is exactly one set of consecutive relays operated, this set consisting of one or two relays. Hint: Design a circuit that is *closed* under the above conditions, and after the ten relay circuits are “strung together” graphically complement the entire network.

11

Number Systems; Adders

Number Systems

A number such as

2,547.16

is not normally thought of as being composed of two 1,000's, five 100's, four 10's, seven 1's, one $\frac{1}{10}$, and six $\frac{1}{100}$'s. However, in a discussion of number systems in general, it will be helpful to think of numbers broken down in this way.

In general, the right-most digit to the left of the radical point represents the number of 1's or B^0 s, where B is the *base* or *radix* of the number system. The next digit to the left represents the number of B^1 s; the next digit to the left represents the number of B^2 s; the next digit to the left represents the number of B^3 s; etc.

The left-most digit to the right of the radical point represents the number of B^{-1} s; the next digit to the right represents the number of B^{-2} s; the next digit to the right represents the number of B^{-3} s; etc.

In the decimal number system, the base is 10. The analysis for the decimal number 2,547.16 is shown below.

2	5	4	7	1	6
$= 2 \times 10^3$	5×10^2	4×10^1	7×10^0	1×10^{-1}	6×10^{-2}
$= 2 \times 1000$	5×100	4×10	7×1	$1 \times \frac{1}{10}$	$6 \times \frac{1}{100}$

2×1000	$= 2000$
5×100	$= 500$
4×10	$= 40$
7×1	$= 7$
$1 \times \frac{1}{10}$	$= \frac{1}{10}$
$6 \times \frac{1}{100}$	$= \frac{6}{100}$

2547 $\frac{16}{100}$ = 2547.16

In a number system to the base B , there are B different symbols, ranging from 0 to $B - 1$. Thus, in the decimal system, base 10, there are ten different symbols, 0 through 9.

The preceding general concepts will now be applied to number systems with other bases. The following number is written in the base 8 number system.

256.71

This number is *not* two hundred fifty-six and seventy-one hundredths. Since it is written in the base 8 number system, it is analyzed as follows:

2	5	6	7	1	(base 8)
$= 2 \times 8^2$	5×8^1	6×8^0	7×8^{-1}	1×8^{-2}	
$= 2 \times 64$	5×8	6×1	$7 \times \frac{1}{8}$	$1 \times \frac{1}{64}$	

2×64	$= 128$
5×8	$= 40$
6×1	$= 6$
$7 \times \frac{1}{8}$	$= \frac{7}{8}$
$1 \times \frac{1}{64}$	$= \frac{1}{64}$

174 $\frac{57}{64}$ (base 10)

256.71 in the base 8 number system represents two 64's, five 8's, six 1's, seven $\frac{1}{8}$'s and one $\frac{1}{64}$. The total $174\frac{57}{64}$ in the base 10 or decimal system, then, is the equivalent of 256.71 in base 8.

The above type of analysis can be used to convert from a number in any base to its decimal equivalent. A convenient method for converting from a decimal number to a number in some other base follows. The decimal

number is separated into two parts: that to the left of the decimal point, and that to the right of the decimal point. Each part is handled in a different way.

The left part of the number is repeatedly divided by the base to which it is to be converted, and the remainders are recorded for each division. This procedure is continued until the quotient 0 is reached. The remainders, reading from the last remainder to the first, represent the left part of the number in the new base.

The right part of the number is repeatedly multiplied by the base to which it is to be converted, and the carries are recorded for each multiplication. This procedure is continued until the product 0 is reached, or until the desired number of places is obtained. The carries, reading from the first carry to the last, represent the right part of the number in the new base.

As an example, the decimal number 174.890625 will be converted to its equivalent in the base 8.

Left part:			Right part:		
		<u>Remainder</u>			<u>Carry</u>
8	<u>174</u>	6		.890625	
8	<u>21</u>	5		<u>× 8</u>	
8	<u>2</u>	2		.125000	
	0			<u>× 8</u>	
				.000000	
					↓ 1

$$174 \text{ (base 10)} = 256 \text{ (base 8)} \qquad .890625 \text{ (base 10)} = .71 \text{ (base 8)}$$

$$174.890625 \text{ (base 10)} = 256.71 \text{ (base 8)}$$

For an example of an approximate conversion, the decimal number .14159 will be converted to its "equivalent" in base 3, correct to four places.

	<u>Carry</u>	
		.14159
		<u>× 3</u>
0		.42477
		<u>× 3</u>
1		.27431
		<u>× 3</u>
0		.82293
		<u>× 3</u>
↓ 2		.46879

$$.14159 \text{ (base 10)} \approx .0102 \text{ (base 3)}$$

A few more examples of other number systems and their conversion to and from decimal are given below for study.

(1)

$$142 \text{ (base 5)} = ? \text{ (base 10)}$$

$$\begin{array}{r} \begin{array}{ccc} 1 & 4 & 2 \\ = 1 \times 5^2 & 4 \times 5^1 & 2 \times 5^0 \\ = 1 \times 25 & 4 \times 5 & 2 \times 1 \\ & 2 \times 1 = 2 \\ & 4 \times 5 = 20 \\ & 1 \times 25 = 25 \\ & \hline & 47 \end{array} \end{array}$$

$$142 \text{ (base 5)} = 47 \text{ (base 10)}$$

(2)

$$47 \text{ (base 10)} = ? \text{ (base 5)}$$

$$\begin{array}{r} 5 \overline{)47} \quad 2 \uparrow \\ 5 \overline{)9} \quad 4 \uparrow \\ 5 \overline{)1} \quad 1 \uparrow \\ \hline 0 \end{array}$$

$$47 \text{ (base 10)} = 142 \text{ (base 5)}$$

(3)

$$201 \text{ (base 3)} = ? \text{ (base 10)}$$

$$\begin{array}{r} \begin{array}{ccc} 2 & 0 & 1 \\ = 2 \times 3^2 & 0 \times 3^1 & 1 \times 3^0 \\ = 2 \times 9 & 0 \times 3 & 1 \times 1 \\ & 1 \times 1 = 1 \\ & 0 \times 3 = 0 \\ & 2 \times 9 = 18 \\ & \hline & 19 \end{array} \end{array}$$

$$201 \text{ (base 3)} = 19 \text{ (base 10)}$$

(4)

$$19 \text{ (base 10)} = ? \text{ (base 3)}$$

$$\begin{array}{r} 3 \overline{)19} \quad 1 \uparrow \\ 3 \overline{)6} \quad 0 \uparrow \\ 3 \overline{)2} \quad 2 \uparrow \\ \hline 0 \end{array}$$

$$19 \text{ (base 10)} = 201 \text{ (base 3)}$$

Question: What is the decimal equivalent of 182 (base 8)?

Answer: There can be no such number as 182 in the base 8; in this base there are only eight allowable symbols, 0 through 7. There is no symbol for 8 in the base 8 number system, any more than there is a symbol for 10 in the decimal system. In the base 8 number system, a 1 in the 8's position represents the value 8, just as in the decimal system, a 1 in the 10's position represents the value 10.

The binary or base 2 number system is of particular importance in computers. Each position in the binary number system has only two possible symbols, 0 or 1. Therefore, binary arithmetic or the storage of binary numbers is a "natural" for circuits which have only two possible states.

Below is an example of binary-to-decimal and decimal-to-binary conversion.

$$101011 \text{ (base 2)} = ? \text{ (base 10)}$$

1	0	1	0	1	1
$= 1 \times 2^5$	0×2^4	1×2^3	0×2^2	1×2^1	1×2^0
$= 1 \times 32$	0×16	1×8	0×4	1×2	1×1
		$1 \times 1 = 1$			
		$1 \times 2 = 2$			
		$0 \times 4 = 0$			
		$1 \times 8 = 8$			
		$0 \times 16 = 0$			
		$1 \times 32 = 32$			
		<u>43</u>			

$$101011 \text{ (base 2)} = 43 \text{ (base 10)}$$

$$43 \text{ (base 10)} = ? \text{ (base 2)}$$

2	<u>43</u>	1	↑
2	<u>21</u>	1	
2	<u>10</u>	0	
2	<u>5</u>	1	
2	<u>2</u>	0	
2	<u>1</u>	1	
	0		

$$43 \text{ (base 10)} = 101011 \text{ (base 2)}$$

Although there are methods for directly converting from one base to another, neither base being decimal, it is convenient to perform the conversion in two steps: from the original base to decimal, and from decimal to the new base.

Following is a table of the values 0 to 20 in the various number systems discussed.

Number systems				
Base 10	Base 8	Base 5	Base 3	Base 2
0	0	0	0	0
1	1	1	1	1
2	2	2	2	10
3	3	3	10	11
4	4	4	11	100
5	5	10	12	101
6	6	11	20	110
7	7	12	21	111
8	10	13	22	1000
9	11	14	100	1001
10	12	20	101	1010
11	13	21	102	1011
12	14	22	110	1100
13	15	23	111	1101
14	16	24	112	1110
15	17	30	120	1111
16	20	31	121	10000
17	21	32	122	10001
18	22	33	200	10010
19	23	34	201	10011
20	24	40	202	10100

Binary Adders

The purpose of this section is to illustrate how switching circuits can be used to perform arithmetic functions, and secondarily, to show how the concept of symmetric functions can be useful in the design of electronic switching circuits.

In the arithmetic addition of two binary digits or “bits,” there are four possible combinations, as shown in Fig. 11-1. A device for adding two bits is called a *half-adder* (Fig. 11-2). A half-adder has two inputs, for the two

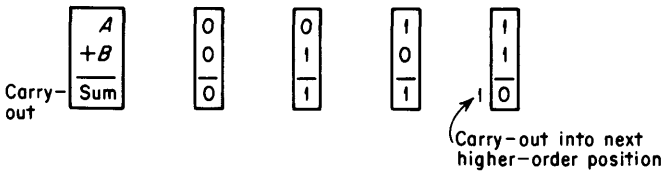


Figure 11-1



Figure 11-2

bits to be added; and two outputs, one for the sum S and one for the “carry-out” C_0 into the next higher-order position.

Observation of the four possible combinations of two bits shows that the sum equals 1 only when $A = 1$ and $B = 0$, or when $A = 0$ and $B = 1$. Furthermore, there is a carry-out into the next higher-order position only when A and B both equal 1.

Boolean expressions for the sum and carry-out outputs of the half-adder can be written as follows:

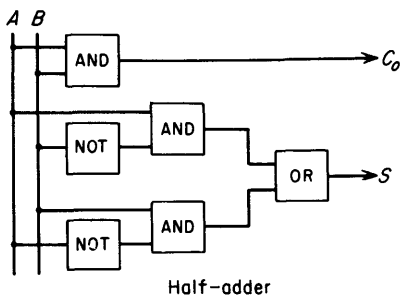
$$S = A\bar{B} + \bar{A}B$$

$$C_0 = AB$$

The logical circuit to implement these expressions is shown in Fig. 11-3. This circuit can be simplified by the manipulation of the Boolean expression for the sum.

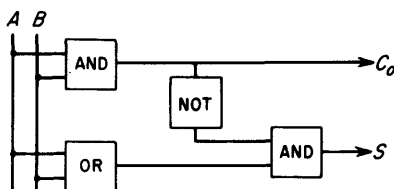
$$\begin{aligned} S &= A\bar{B} + \bar{A}B \\ &= (A + B)(\bar{A} + \bar{B}) \\ &= (A + B)(\overline{AB}) \end{aligned}$$

The simplified half-adder is shown in Fig. 11-4.



Half-adder

Figure 11-3



Simplified half-adder

Figure 11-4

When two bits, A and B , are added in a position, and there is a “carry-in” C_I from the next lower-order position, three bits in all must be added. In the addition of three bits, there are eight possible combinations (Fig. 11-5).

C_I	0	0	0	1	0	1	1	1
A	0	0	1	0	1	0	1	1
B	0	1	0	0	1	1	0	1
C_0	0	1	1	1	0	1	0	1
S	0	1	1	1	0	1	0	1

Figure 11-5

Figure 11-6 shows an example of binary addition. A device for adding three bits is called a *full-adder* (Fig. 11-7). A full-adder has three inputs: A , B , and C_I , and two outputs: S and C_o .

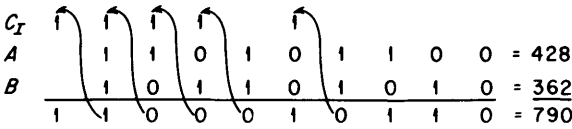


Figure 11-6

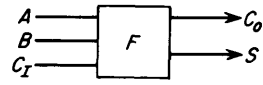


Figure 11-7

In the addition of three bits, the sum equals 1 only when exactly one or three of the bits equal 1. The sum can be expressed as

$$S = A\bar{B}\bar{C}_I + \bar{A}B\bar{C}_I + \bar{A}\bar{B}C_I + ABC_I$$

or in the symmetric notation as

$$S = S_{1,3}^3 ABC_I$$

The carry-out into the next higher-order position equals 1 only when exactly two or three of the bits equal 1. The carry-out can be expressed as

$$\begin{aligned} C_o &= A\bar{B}C_I + \bar{A}B C_I + \bar{A}\bar{B}C_I + ABC_I \\ &= AB + AC_I + BC_I \end{aligned}$$

or in the symmetric notation as

$$C_o = S_{2,3}^3 ABC_I$$

By the intuitive manipulation of the above expressions, particularly the symmetric notations, some economical full-adders can be obtained. If the expression for C_o is factored, the resulting expression will be found useful for implementation:

$$C_o = AB + C_I(A + B)$$

The direct implementation of the expression for S is not very economical. However, realizing that a $S_{0,1}^3 ABC_I$ circuit can be obtained simply by the complementation of the C_o circuit, and that $S_{1,2,3}^3 ABC_I$ and $S_{1,2,3}^3 ABC_I$ circuits are easily implemented by AND and OR circuits respectively, the following useful relationships can be utilized:

$$\begin{aligned} (1) \quad S &= (S_{0,1}^3 ABC_I \cdot S_{1,2,3}^3 ABC_I) + S_{1,2,3}^3 ABC_I \\ &= S_{1,2,3}^3 ABC_I + S_{1,2,3}^3 ABC_I \\ &= S_{1,3}^3 ABC_I \end{aligned}$$

$$\begin{aligned} (2) \quad S &= (S_{0,1}^3 ABC_I + S_{1,2,3}^3 ABC_I) S_{1,2,3}^3 ABC_I \\ &= (S_{0,1,3}^3 ABC_I) (S_{1,2,3}^3 ABC_I) \\ &= S_{1,3}^3 ABC_I \end{aligned}$$

Using the first of the two relationships above, the full-adder circuit in Fig. 11-8 can be obtained. A full-adder can also be constructed with two half-adders and an OR circuit (Fig. 11-9). Some key points in the circuit have been defined to aid in analysis.

The general structure of a 4-position binary adder would appear as in Fig. 11-10.

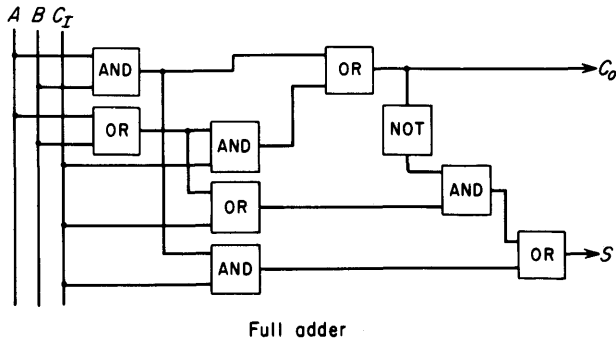


Figure 11-8

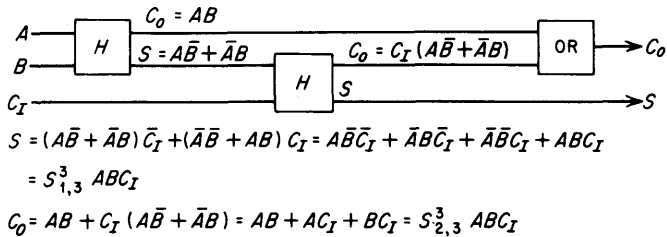


Figure 11-9

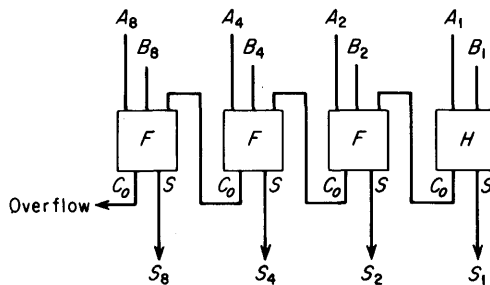


Figure 11-10

Binary-Coded-Decimal Adder

To further illustrate how logical circuits can be used to implement arithmetic functions, a binary-coded-decimal (BCD) adder will be discussed.

The BCD code differs from the straight binary number representation in that in the BCD code *each decimal digit is binary-coded*. For example, the decimal number 13, in straight binary number representation, is

1101

whereas in the BCD code, 13 is represented by

0001 0011

the decimal digits 1 and 3 each being binary-coded.

In the BCD code, the highest allowable binary representation is 1001 (9). Therefore, the highest two numbers that may be added are 1001 + 1001 (9 + 9). Also, there may be a carry-in from the next lower-order position. Thus, the maximum sum that can occur is 10011 (19). However, when the sum exceeds 1001 (9), a correction must be made, as indicated in the following table.

	<i>Uncorrected Sum</i>		<i>Corrected Sum</i>
	<i>C₀ 8421</i>		<i>C₀ 8421</i>
0	0000	<div>↑ No Correction Necessary ↓</div>	0000
1	0001		0001
2	0010		0010
3	0011		0011
4	0100		0100
5	0101		0101
6	0110		0110
7	0111		0111
8	1000		1000
9	1001		1001
10	1010		1 0000
11	1011		1 0001
12	1100		1 0010
13	1101		1 0011
14	1110		1 0100
15	1111		1 0101
16	1 0000		1 0110
17	1 0001		1 0111
18	1 0010		1 1000
19	1 0011		1 1001

Analysis of the table shows that the correction should be made when the uncorrected sum contains an 8 and 2, or an 8 and 4, or when there is a carry-out from the 8's position. Analysis of the table also shows that the corrected sum can be obtained by adding 0110 (6). This is numerically equivalent to subtracting 1010 (10) and adding 1 0000 (16) (generating a carry-out to the next higher-order position).

The circuit in Fig. 11-11 illustrates one decimal position of a BCD adder.

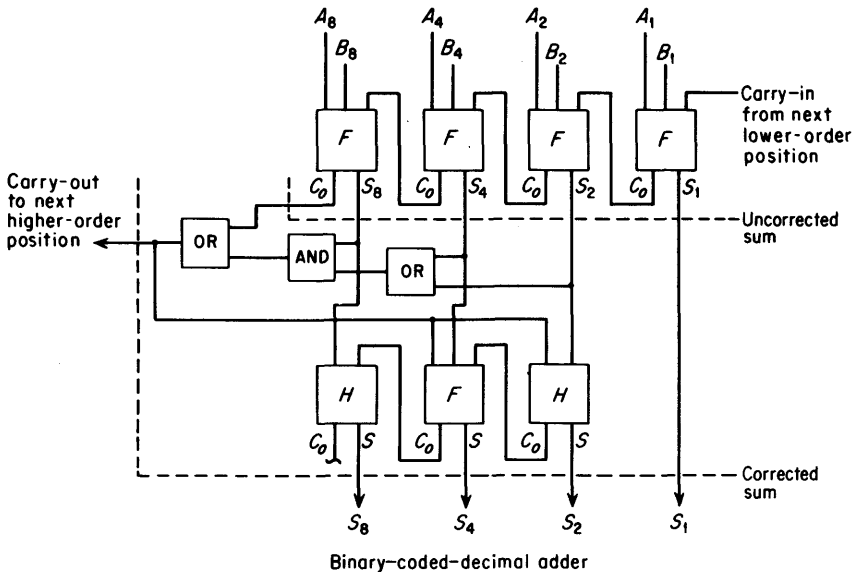


Figure 11-11

PROBLEMS

1. Convert 111011 (base 2) to base 3.
2. Convert 2601 (base 7) to base 6.
- *3. Convert 3333 (base 6) to base 7.
4. Convert 13.8125 (base 10) to base 2.
- *5. Convert 49.296875 (base 10) to base 4.

12

Codes, Error Detection, Error Correction

In this chapter, some of the more popular codes used for data representation will be discussed. These codes are used for such things as arithmetic processes and storage and transmission of information. For example, instead of a decimal 6 being represented by a signal on one of ten lines (Fig. 12-1) or by one of ten timed signals (Fig. 12-2) the 6 may be binary-coded as 0110, and represented with only four lines (Fig. 12-3) or with only four timed signals (Fig. 12-4).

Nonchecking Numeric Codes

First, various schemes for coding the ten decimal digits, 0 through 9, will be examined.

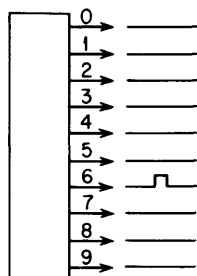


Figure 12-1

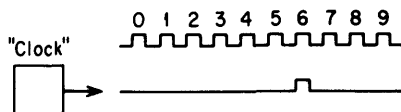


Figure 12-2

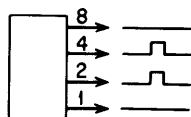


Figure 12-3

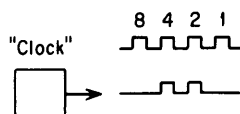


Figure 12-4

BCD Code

One of the most logical codes for representing the decimal digits is the binary-coded-decimal or BCD code. Four "bits" (binary digits) are required to code the ten decimal digits.

	8	4	2	1
BCD code {	0	0	0	0
	1	0	0	1
	2	0	0	1
	3	0	0	1
	4	0	1	0
	5	0	1	0
	6	0	1	1
	7	0	1	1
	8	1	0	0
	9	1	0	0

Excess-3 Code

The BCD code may be thought of as utilizing the first ten of the sixteen possible combinations of four bits. Another code, which utilizes the middle ten of these sixteen combinations, is called the Excess-3 code. Each coded

character is the binary equivalent of the represented decimal number *plus three*.

A property of the Excess-3 code that makes it useful in arithmetic is that the 9's complement of a decimal digit may be obtained by complementing all bits. For example, the coding for the decimal digit 1 is 0100. The 9's complement of 1 is 8, which, in the Excess-3 code, is 1011. Complementing all bits of 0100 results in 1011.

	8	4	2	1
	0	0	0	0
	0	0	0	1
	0	0	1	0
	0	0	1	1
	0	1	0	0
	0	1	0	1
	0	1	1	0
	0	1	1	1
	1	0	0	0
	1	0	0	1
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Cyclic Codes

Sometimes it is desirable to have a code in which successive coded characters differ in only one bit position. Such codes are called cyclic codes, and they are particularly useful in analog-digital systems.

One type of cyclic code is the reflected code. A reflected binary code for sixteen decimal digits follows. This code is also known as the Gray code.

Note that except for the high-order position, all columns are "reflected" about the mid point; in the high-order position, the top half is all 0's and the bottom half all 1's. This pattern can be used for a reflected binary code of any number of bits. A reflected code for three bit positions is enclosed by dotted lines for illustration.

Error Detection and Correction

None of the codes discussed so far can be error-checked. If bits become erroneously changed, say because of circuit failure, there would be no general way to detect the error because in these codes there are cases of two coded characters differing in only one bit position. If even only a single bit in a character became erroneously changed, another valid character could result, and there would be no way of knowing that the resultant character was not the intended one. The characteristics of error-detecting and error-correcting codes will now be discussed.

The *distance* between two coded characters is the number of bits that must change in one character so that the other character results. For example, the distance between the coded characters 0011 and 1000 is three, since three bits must change to transform one of the characters to the other.

The *minimum distance* of a code is the minimum number of bits that must change in a coded character so that another valid character of the code will result.

In all of the codes discussed so far, the minimum distance was 1: there was at least one case in each code in which a coded character could be changed to another by changing only one bit.

The relationship between the minimum distance of a code and the amount of error detection or correction possible is as follows:

$$M - 1 = C + D, \quad \text{where} \quad C \leq D$$

M = minimum distance of a code

C = number of bits in error that can be corrected

D = number of bits in error that can be detected

Since no error can be corrected without being detected, C cannot be greater than D . All possible values for C and D for values of M up to six are tabulated on the next page.

An error-detection code is defined according to one less than the minimum error it will not *always* detect. Thus, if a code detects *all* single, double and triple errors, and *some* or *no* quadruple errors, it is called a triple-error detecting code. This would still be so even if the code detected *all* quintuple errors. An error correction code is defined in the same manner, that is, according to one less than the minimum error it will not *always* correct.

The relationship between the minimum distance of a code and the amount of error correction or detection possible may be more graphically pictured if a "table-lookup" error detection system is considered. All the valid characters in the code are stored in the table. Each coded character to be checked is compared with the characters in the table. If a character in

M	C	D
1	0	0
2	0	1
3	0	2
	1	1
4	0	3
	1	2
5	0	4
	1	3
	2	2
6	0	5
	1	4
	2	3

the table is found to match exactly, it is assumed that no error has occurred; if no character matches exactly, an error has been detected. Whether or not the error can be corrected depends upon the minimum distance of the code, as will be seen.

In codes with a minimum distance of one, where two valid characters may differ in only one bit position, a single error in a character could make that character appear like another valid character in the code. This other valid character would be found in the table, and it would be falsely assumed that no error had occurred. Thus, in codes having a minimum distance of one, single errors can fail to be detected. Of course, if single errors can be undetected, multiple errors can be undetected also.

In codes with a minimum distance of two, all coded characters must differ in at least two bit positions. If there is a single bit error in a character, the character cannot possibly match any of those in the table; therefore, all single errors will be detected. Codes with a minimum distance of two are called single-error detecting codes. Errors in two or more bits might make the character match exactly some other valid character in the table, and therefore these errors would not be detected.

In codes with a minimum distance of three, all coded characters must differ in at least three bit positions. A character with a single or double bit failure cannot match any character in the table; therefore, all single and double errors will be detected. Errors in three or more bits can result in another valid character and therefore these errors cannot be detected.

Minimum distance three codes can be used for single-error correction. The key to error correction is that it must be possible to *locate* the bit or bits in error. If a single error occurs in a minimum distance three code, the resulting character will not match exactly any character in the table,

but it will come *within one bit* of matching the correct character. It will not come within one bit of matching any other character. To accomplish the correction, the one bit that does not match is changed.

In any code that can be used for correction, correction is “bought” at the expense of detection. If a minimum distance three code is used for correction, and a double error occurs, the resulting character may come within a single bit of matching some other character in the table. Since there is no way of knowing that a double error has occurred, it would be assumed that the single bit was in error, and this bit would be erroneously “corrected.” Thus, the error would be compounded, and an incorrect character would result. Minimum distance three codes thus will not detect double errors if they are used to correct single errors.

Summarizing, if minimum distance three codes are used for correction, the location of one bit in error can be determined and the error corrected. Errors in two or more bits can appear to the error correction system as a single error, and an erroneous correction (undetected error) can result. Minimum distance three codes are often referred to as single-error correcting codes.

The characters in minimum distance four codes differ in at least four bit positions. Single, double, and triple errors can be detected with these codes, since the resulting character cannot match any of those in the table. Errors in four or more bit positions can result in a character that matches some other valid character in the table, and thus these errors cannot be detected.

Instead of minimum distance four codes being used for triple-error detection, they can be used for single-error correction with double-error detection. If a single error occurs, the resulting character will not match exactly any character in the table, but it will come within one bit of matching the correct character. It will differ from all other characters in the table by at least three bits. The correction is made by changing the one bit that does not match.

A character with a double error will come within two bits of matching the correct character of the table, but it may also come within two bits of matching an incorrect character. There is, therefore, no way of knowing which bits are actually in error and so no attempt is made to correct double errors; they are simply detected.

If a triple error occurs in a character, the resultant character will differ from the correct one in the table in three bit positions, but it may differ from some other character in the table in only one bit position. This one bit would thus be erroneously “corrected,” and an incorrect character would result.

Summarizing, if minimum distance four codes are used for correction, the location of one bit in error can be determined and the error corrected.

Double errors can be detected but their location cannot be determined for correction. Errors in three or more bit positions can appear to the error correction system as a single error, and an erroneous correction (undetected error) can result.

Minimum distance four codes are often referred to as single-error correcting, double-error detecting codes.

The table lookup system was used as an aid in learning the concept of minimum distance as it relates to error detection and correction. In practice there are many schemes for accomplishing error detection and correction. As other codes are now examined, it will be seen how some of these schemes work.

Single-Error Detection—Minimum Distance

Two Codes

A single-error detecting code can be obtained by adding a redundant bit to a nonchecking code. The redundant bit can be added to each character in such a way as to make the number of 1 bits in the character even. If this is done, the code is referred to as an “even parity” or “even redundancy” code. The redundant bit may instead be added to each character so as to make the number of 1 bits in the character odd, giving an “odd parity” or “odd redundancy” code. Following are examples of both types of parity codes.

8	4	2	1	R		8	4	2	1	R
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	1	0	0	0	1	0
0	0	1	0	1	2	0	0	1	0	0
0	0	1	1	0	3	0	0	1	1	1
0	1	0	0	1	4	0	1	0	0	0
0	1	0	1	0	5	0	1	0	1	1
0	1	1	0	0	6	0	1	1	0	1
0	1	1	1	1	7	0	1	1	1	0
1	0	0	0	1	8	1	0	0	0	0
1	0	0	1	0	9	1	0	0	1	1

Even Parity BCD Code

Odd Parity BCD Code

The odd parity *BCD* code is sometimes preferred over the even parity *BCD* code because an all-0 character is frequently undesirable: if a gross circuit failure can change a character to all 0's, it is desirable that an all-0 character not be one of the valid characters in the code. However, a modification is frequently made in which the binary 1010 is assigned to the

decimal 0; thus, the even parity character 10100 rather than 00000 represents the decimal 0.

Characters in these codes are checked for the proper parity. If a single error occurs, it will be detected because the character will have the wrong parity. Double errors will not be detected since the parity will check correctly.

Another class of codes are the *fixed-bit* or *m-out-of-n* codes. In these codes there are n bits per character, of which m bits are 1's. Such a code suitable for representing the ten decimal digits is the 2-out-of-5 code, there being exactly ten combinations of five things taken two at a time (${}_5C_2$).

While any assignment of the ten 2-out-of-5 combinations to the ten decimal digits could be made, there are some that are more convenient to remember. It is not possible to correctly "weight" all ten combinations, but it is possible to properly weight nine of them. Two such weightings are shown: the 01247 code and the 01236 code.

2-out-of-5 Codes

0	1	2	4	7		0	1	2	3	6
0	0	0	1	1	0	0	1	1	0	0
1	1	0	0	0	1	1	1	0	0	0
1	0	1	0	0	2	1	0	1	0	0
0	1	1	0	0	3	1	0	0	1	0
1	0	0	1	0	4	0	1	0	1	0
0	1	0	1	0	5	0	0	1	1	0
0	0	1	1	0	6	1	0	0	0	1
1	0	0	0	1	7	0	1	0	0	1
0	1	0	0	1	8	0	0	1	0	1
0	0	1	0	1	9	0	0	0	1	1

Only the decimal 0 is improperly weighted in both codes.

There are two other 2-out-of-5 codes that weight nine of the ten combinations correctly, but both of these involve negative weights:

- 1, 2, 3, 4, 5
- 2, 1, 3, 4, 5

A popular 2-out-of-5 code that weights eight of the ten combinations properly is the 84210 code. All combinations are weighted correctly except the decimal 0 and 7; the 8—2 combination is used for the decimal 0, and the 8—4 combination is used for the decimal 7. This code is not very different from the even parity BCD code and very little logical circuitry is needed to convert from one of these codes to the other.

Characters in fixed bit codes are checked for the correct number of 1-bits.

Single errors will be detected since the number of 1-bits in the character will be one too many or one too few. Double errors involving two 1's or two 0's will also be detected, but double errors in which a 0 becomes a 1, and a 1 becomes a 0, will not be detected.

Another popular fixed bit code is the biquinary code. This is a seven-bit code made up of a 1-out-of-2 group and a 1-out-of-5 group. Again, there are ten possible combinations. Two possible weightings for this code are shown; the second one is sometimes called the "quibinary code" to differentiate it from the first one.

Biquinary Codes

0 5	0 1 2 3 4		0 1	0 2 4 6 8
1 0	1 0 0 0 0	0	1 0	1 0 0 0 0
1 0	0 1 0 0 0	1	0 1	1 0 0 0 0
1 0	0 0 1 0 0	2	1 0	0 1 0 0 0
1 0	0 0 0 1 0	3	0 1	0 1 0 0 0
1 0	0 0 0 0 1	4	1 0	0 0 1 0 0
0 1	1 0 0 0 0	5	0 1	0 0 1 0 0
0 1	0 1 0 0 0	6	1 0	0 0 0 1 0
0 1	0 0 1 0 0	7	0 1	0 0 0 1 0
0 1	0 0 0 1 0	8	1 0	0 0 0 0 1
0 1	0 0 0 0 1	9	0 1	0 0 0 0 1

An advantage of these codes is that the circuitry to perform arithmetic operations is quite economical. The quibinary code has the advantage of more economical conversion to and from the BCD code.

Single-Error Correction—Minimum Distance

Three Codes

The construction and operation of a *Hamming code* will be used as an example in the study of single-error correcting codes.

First of all, in determining how many bits per character are required, the bit positions are numbered sequentially from left to right as 1, 2, 3, etc. The positions that are a power of two, that is positions 1, 2, 4, 8, 16, etc., are reserved for check bits. All other bit positions may then contain information bits.

If a single-error correcting numeric code is required, and the four-bit BCD code is used for the information, seven bits in all would be required: positions 1, 2, and 4 for check bits, and positions 3, 5, 6, and 7 for the four information bits. These seven bits can be labeled as follows:

1	2	3	4	5	6	7
<hr/>						
C_1	C_2	8	C_4	4	2	1

The values of the check bits C_1 , C_2 , and C_4 , for each coded character, are determined as follows:

- C_1 is chosen so as to establish even parity for positions 1, 3, 5, and 7.
- C_2 is chosen so as to establish even parity for positions 2, 3, 6, and 7.
- C_4 is chosen so as to establish even parity for positions 4, 5, 6, and 7.

This pattern may be more obvious if the position locations are written in binary.

(C_1)	1	0	1	0	1	0	1
(C_2)	0	1	1	0	0	1	1
(C_4)	0	0	0	1	1	1	1
<hr/>							
	1	2	3	4	5	6	7
<hr/>							
	C_1	C_2	8	C_4	4	2	1

For an example, the check bits for the character for the decimal 9 will be generated.

1	2	3	4	5	6	7
<hr/>						
C_1	C_2	8	C_4	4	2	1
<hr/>						
1		0	0	1		

C_1 must be chosen so as to establish even parity for positions 1, 3, 5, and 7; therefore, C_1 must be a 0.

1	2	3	4	5	6	7
<hr/>						
C_1	C_2	8	C_4	4	2	1
<hr/>						
0	1	0	0	1		

C_2 must be chosen so as to establish even parity for positions 2, 3, 6, and 7; C_2 must also be a 0.

1	2	3	4	5	6	7
<hr/>						
C_1	C_2	8	C_4	4	2	1
<hr/>						
0	0	1		0	0	1

C_4 must be chosen so as to establish even parity for positions 4, 5, 6, and 7; C_4 must therefore be a 1.

1	2	3	4	5	6	7
C_1	C_2	8	C_4	4	2	1
0	0	1	1	0	0	1

The coded character for the decimal 9 is therefore

0011001

To illustrate how a single error in this coded character can be detected and corrected, an error will be "made" in position 6.

1	2	3	4	5	6	7
0	0	1	1	0	1	1

↑

The three parity checks, involving C_1 , C_2 , and C_4 , are applied to the character. Based on the outcome of these checks, a binary number is developed, the C_1 , C_2 , and C_4 checks corresponding respectively to the 1, 2, and 4 positions of the binary number. If the check shows even (correct) parity, a 0 is entered in the corresponding position of the binary number; if the check shows odd (incorrect) parity, a 1 is entered. The resulting binary number indicates the position in error; to correct the error, the bit in the position indicated is changed.

In this example, the three parity checks are as follows:

C_1 :	0	0	1	1	0	1	1	even
C_2 :	0	0	1	1	0	1	1	odd
C_4 :	0	0	1	1	0	1	1	odd

1 1 0 = 6

The C_1 parity check shows even parity, while the C_2 and C_4 parity checks show odd parity. The resulting binary number, $110 = 6$, indicates that position 6 is in error. To correct the error, the bit in position 6 is changed from a 1 to a 0. Study of the construction of this code will show that the position of any bit in error is uniquely identified by the outcome of the parity checks. If the resultant binary number is zero (000), no error is indicated.

Only single errors are detected and corrected with this code. Errors in

two bit positions will appear to the error correction system as a single error, and a false correction will be made. Triple errors may also appear as single errors and be falsely corrected, or they may “cancel out” and appear as no error.

If this code is used for detection only, all single and double errors will be detected; the error detection system checks only for the occurrence of an error, but does not try to identify a position in error and correct it.

The Hamming code is by no means the only type allowing single-error correction. As long as the coded characters are chosen so that all pairs of characters differ in *at least* three bit positions, single-error correction can be accomplished.

Single-Error Correction with Double-Error Detection—Minimum Distance Four Codes

Hamming single-error correcting codes can be extended into Hamming single-error correcting double-error detecting codes simply by the addition of one more bit establishing even parity over the entire coded character.

For example, taking the seven-bit character for the decimal 9, if an eighth bit is added to establish even parity over the entire character, this bit must be a 1, and the resulting character for the 9 is

00110011

Four parity checks are made on the character: the C_1 , C_2 , and C_4 checks, and the over-all parity check which can be called the P check. Any single error will be indicated by the P check showing odd parity. If the single error occurs in any of the first seven-bit positions, it will show up in some combination of the C_1 , C_2 , and C_4 checks, which will indicate the position in error. If the single error is in the eighth bit, the absence of any error indication by the C_1 , C_2 , and C_4 checks indicates that the bit in error is the eighth bit. The P check showing odd parity thus indicates that a single error has occurred, and that a correction should be made.

If a double error occurs, the P check will show even parity. Even though the C_1 , C_2 , and C_4 checks indicate some position in error, the P check showing even parity indicates that a double error has occurred, and that no correction should be made.

Alphanumeric Codes

“Alphanumeric” codes are those containing enough coded characters to code the ten decimal digits, the twenty-six letters of the alphabet, and

often special symbols also. A few examples of such codes will be described.

The BCD code can be expanded into a six-bit code giving 64 possible coded characters, satisfying the requirement for an alphanumeric code. To make such a code into a single-error detecting code, a parity bit can be added.

There are various *m-out-of-n* codes used for alphanumeric information. For instance, 3-out-of-8 codes (56 characters) and 4-out-of-8 codes (70 characters) are used.

Alphanumeric Hamming single-error correcting codes require ten bits in all, six information bits and four check bits, the check bits occupying positions 1, 2, 4, and 8. By adding an eleventh bit to this code, an alphanumeric Hamming single-error correcting double-error detecting code is obtained.

Cross-Parity

Sometimes a check is associated with an entire block of characters. For instance, at the end of a block of even parity BCD characters, an entire redundant character is added, the bits in this character being chosen so as to establish even parity in each "channel," that is, the 8-bit channel, the 4-bit channel, the 2-bit channel, etc.

Parity checks on this block of information are made in two directions: "vertically" for each character and "horizontally" for each channel. This code will detect all single, double and triple errors, or it may be used as a single-error correcting double-error detecting code.

EXAMPLE:

		<i>Characters</i>								
		9	5	7	1	4	3	8	2	<i>R</i>
8	1	0	0	0	0	0	1	0	0	
4	0	1	1	0	1	0	0	0	1	
2	0	0	1	0	0	1	0	1	1	
1	1	1	1	1	0	1	0	0	1	
<i>R</i>	0	0	1	1	1	0	1	1	1	

Suppose that an error occurs in the "1" bit of the "3" character.

Characters

	9	5	7	1	4	3	8	2	R
8	1	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	0	0	1
2	0	0	1	0	0	1	0	1	1
1	1	1	1	1	0	0	0	0	1
R	0	0	1	1	1	0	1	1	1

^

The odd vertical parity on the “3” character and the odd horizontal parity on the “1” channel locate the single error for correction.

PROBLEM

1. Each digit of a 4-digit decimal number is encoded in a single-error correcting double-error detecting Hamming code. The coded information is received high-order digit first as follows:

	1	2	3	4	5	6	7	8
	C_1	C_2	8	C_4	4	2	1	P
? →	0	1	0	1	1	1	0	1
? →	1	1	0	0	0	1	1	1
? →	1	0	0	1	1	0	0	0
? →	0	1	0	0	1	1	1	0

Correct and decode:

1	2	3	4	5	6	7	8
C_1	C_2	8	C_4	4	2	1	P

→
→
→
→

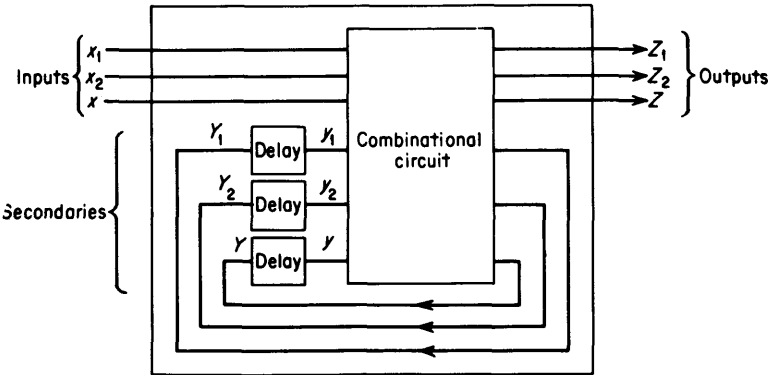
13

Sequential Circuits I

The circuits that have been discussed so far are called *combinational circuits*. In combinational circuits, the outputs are functions solely of the inputs: for a particular input combination there will either always be an output or else there will never be an output.

The rest of this book discusses *sequential circuits* (Fig. 13-1). In sequential circuits, time is an element, that is, sequential circuits have a memory, the outputs being functions not only of the present inputs but also of past circuit states.

Sequential circuit inputs are also called *primaries*, and their states are represented by x 's. The states of sequential circuit outputs are represented by Z 's. The memory characteristic of sequential circuits is realized by secondary circuit components or *secondaries*. A property of the secondaries is that there is a time delay between their excitation and their resulting change of state. The *states* of the secondaries are represented by y 's. Their *excitations* are represented by Y 's. The next state of a secondary will be the same as its present excitation; that is, a y will, after the time delay, become



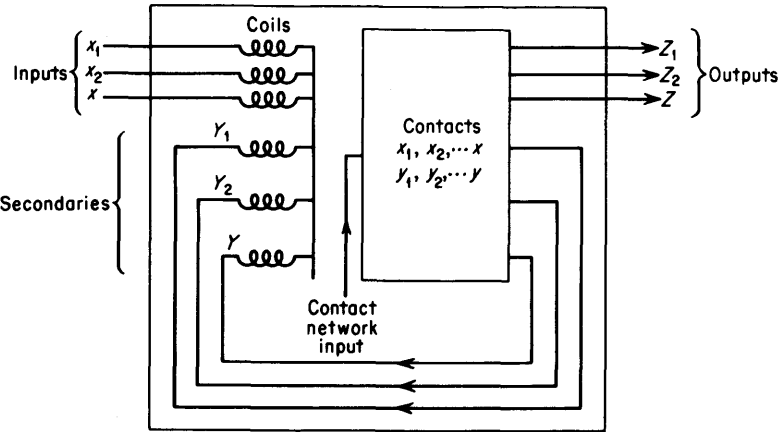
Schematic diagram of sequential switching circuit

Figure 13-1

the same as the corresponding present Y . Thus, the Y 's also represent the *next states* of the secondaries.

In electronic sequential circuits, the secondaries are *feedback paths*—paths leaving the combinational circuit and feeding back into it. The delay may be inserted or may be inherent in the feedback paths. Also, all feedback paths must have *gain*, so that the circuits involved are self-sustaining; therefore, amplifiers may have to be inserted in the feedback paths if inherent amplification is not already present.

In relay sequential circuits (Fig. 13-2), the secondaries are relays. The delay is inherent in the operate and release time of the secondary relays.



Schematic diagram of relay sequential switching circuit

Figure 13-2

The state of a secondary relay is described by the operation or inoperation of the contacts. The excitation of a secondary relay is described by the energization or deenergization of the coil.

Primary relays are under the direct control of the inputs, and are used to make a multiplicity of contacts available for switching. Therefore, the states of the primary relays are considered as equivalent to the states of the corresponding inputs.

Concept of Stability

When the excitation of a secondary is the same as the present state, that is, $Y = y$, the next state will be the same as the present state, and since the secondary will not change state, it is said to be *stable*.

When the excitation of a secondary is not the same as the present state, that is, $Y \neq y$, the next state will not be the same as the present state, and since the secondary will change state, it is said to be *unstable*.

If $y = 0$ and $Y = 0$, the next state will be 0, the secondary state will not change, and the secondary is stable.

If $y = 1$ and $Y = 1$, the next state will be 1, the secondary state will not change, and the secondary is stable.

If $y = 0$ and $Y = 1$, the next state will be 1, the secondary state will change, and the secondary is unstable.

If $y = 1$ and $Y = 0$, the next state will be 0, the secondary state will change, and the secondary is unstable.

The concept of stability can be easily visualized in terms of relay operation. Consider the coil and a normally-open contact of a secondary relay (Fig. 13-3). The following assignment is made:

$y = 0$: contact unoperated

$y = 1$: contact operated

$Y = 0$: coil deenergized

$Y = 1$: coil energized



Figure 13-3

Assume, to start with, that the switch is open, the coil deenergized, and the contact unoperated. At this time,

$y = 0$ $Y = 0$ secondary is stable

Now assume that the switch is moved to the closed position. For a brief period of time, the coil is energized, but the contact is still unoperated. During this time,

$$y = 0 \quad Y = 1 \quad \text{secondary is unstable}$$

This unstable condition will terminate when the contact operates, and at this time,

$$y = 1 \quad Y = 1 \quad \text{secondary is stable}$$

Now assume that the switch is returned to the open position. For a brief period of time, the coil is deenergized, but the contact is still operated. During this time,

$$y = 1 \quad Y = 0 \quad \text{secondary is unstable}$$

This unstable condition will end when the contact returns to its normal state, and at this time, again,

$$y = 0 \quad Y = 0 \quad \text{secondary is stable}$$

Basic Sequential Circuit Operation

Refer to the two preceding schematic diagrams of sequential switching circuits (Figs. 13-1 and 13-2). The states of the outputs (Z 's) and the secondary excitations (Y 's) are functions of the states of the inputs (x 's) and the secondary states (y 's). Note that the secondaries enter into their own control.

Assume a sequential circuit to be stable. A change in input states (x 's) may or may not cause a change in the secondary excitations (Y 's). If a change in Y 's does occur, the corresponding secondary states (y 's) will, after a delay, also change. If following the change in y 's, the circuit is stable, no further change will occur. If the circuit is unstable, another change in Y 's will occur, followed by a change in the corresponding y 's. These changes will continue until a stable circuit is reached.

Successive input changes must be spaced far enough apart in time so that sufficient time is allowed for the completion of all required secondary action before another input change occurs.

Intuitive Approach to Sequential Circuit Synthesis

Before the formal method of sequential circuit design is examined, an intuitive approach will be discussed. For the more simple sequential circuit requirements, the intuitive approach can be satisfactorily applied. For the more complex requirements, the intuitive approach can become more

difficult to apply, and an optimum solution may not be obtained or recognized except by the formal method of synthesis.

In the first example to be considered, there is only one sequence that can occur. A circuit is to have two inputs, x_1 and x_2 , and one output, Z . Starting from a condition of both inputs off, x_1 will turn on first. While x_1 is still on, x_2 will turn on, and then later turn off. Following this, x_1 will turn off, the inputs returning to their original state. The output, Z , is to turn on when x_2 turns off, and the output is to remain on until x_1 turns off. This sequence can be shown in a *timing chart* (Fig. 13-4).

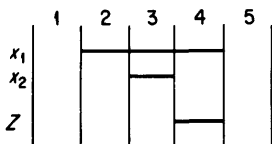


Figure 13-4

In a timing chart such as Fig. 13-4, a vertical division is allotted to each input state in the sequence. Each division actually represents some interval of time during which the corresponding input state exists. Although the duration of these time intervals may vary, it is convenient on the timing chart to make all divisions of equal length. A horizontal line is drawn

through those intervals in which the corresponding input or output is on; the absence of this line indicates the off condition. The intervals are numbered sequentially for reference.

Intervals 2 and 4 on the timing chart of Fig. 13-4 indicate the sequential aspect of the circuit requirement: the same input conditions exist during intervals 2 and 4; however, the output requirements are not the same, no output being desired during interval 2 but an output being desired during interval 4. When two or more intervals have the same input conditions but different output conditions, secondaries must be used to differentiate one interval from the other. In the example, therefore, secondaries must be used to differentiate between intervals 2 and 4.

The intuitive approach is generally as follows:

- (1) Determine which intervals must be differentiated from one another.
- (2) Devise an operating sequence of secondaries that will accomplish this differentiation.
- (3) Design the secondary excitation circuits.
- (4) Design the output circuits.

It is generally desirable to minimize the number of secondaries required (step 2), and to minimize the secondary excitation circuits (step 3), and the output circuits (step 4). It is, of course, most desirable to minimize the total circuit, and towards this end one should be aware that the secondary excitation circuits and output circuits may be able to share logic blocks or contacts in common.

Returning to the example, how can secondaries differentiate intervals 2 and 4? Inspection of the timing chart shows that if a secondary is off during interval 2 and on during interval 4, the two intervals will be differentiated.

To accomplish this secondary action, the secondary must turn on in interval 3, and turn off in interval 5. (A secondary that was on during interval 2 and off during interval 4 would also accomplish the differentiation. However, this secondary action would necessitate turning on the secondary in interval 1, and it is generally common design practice not to turn on any secondaries in interval 1, the “normal” or “power-on” interval.) Only one secondary is therefore required, and its operating sequence is incorporated in the timing chart (Fig. 13-5). The secondary excitation Y is not normally shown on a timing chart and is shown here only for discussion purposes. Also, the time intervals A , B , C , and D are labeled for reference purposes only. Note the time delay between the excitation and change of state of the secondary.

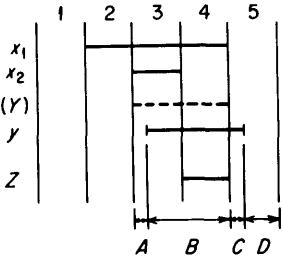


Figure 13-5

To review the concept of stability, assume relay implementation. The energization of the secondary relay is indicated by Y on the timing chart, and the operation of the relay is indicated by y .

During the time interval labeled A , the relay is energized but unoperated ($y = 0$, $Y = 1$, secondary unstable). During time interval B , the relay is energized and operated ($y = 1$, $Y = 1$, secondary stable). During time interval C , the relay is deenergized but operated ($y = 1$, $Y = 0$, secondary unstable). At all other times, the relay is deenergized and unoperated ($y = 0$, $Y = 0$, secondary stable).

Now that it has been determined that one secondary is required, and the operating sequence of the secondary has been prescribed, the next step is the design of the secondary excitation circuit. Although, in such a simple problem, the circuit can be designed by inspection of the timing chart, a map will be used for later comparison with the formal method of synthesis. The map for the secondary excitation Y is shown in Fig. 13-6. The accompanying table, showing the states of the variables, their corresponding timing chart intervals, and excitation Y , is given for reference.

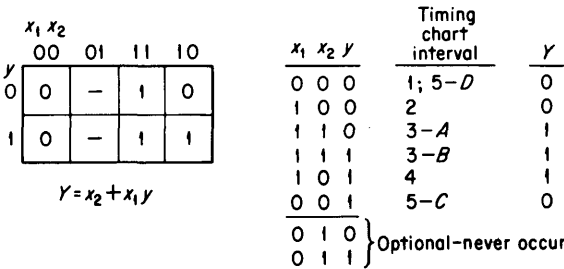


Figure 13-6

The secondary excitation circuit is shown in Fig. 13-7, with both electronic and relay implementation. Note that the secondary enters into its own control. Next, the output circuit is designed. The output or Z-map, accompanying table, and output circuit are shown in Fig. 13-8.

Note, in the total circuit (Fig. 13-9), how the secondary excitation circuit and output circuit have been combined in both implementations.

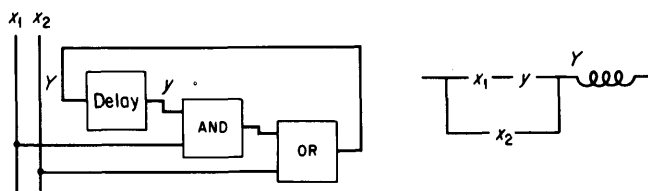


Figure 13-7

$x_1 x_2$							
y	00	01	11	10	$x_1 x_2 y$	Timing chart interval	Z
0	0	—	0	0	0 0 0	1; 5-D	0
					1 0 0	2	0
1	0	—	0	1	1 1 0	3-A	0
					1 1 1	3-B	0
					1 0 1	4	1
					0 0 1	5-C	0
					0 1 0	Optional—never occur	
					0 1 1		

$$Z = x_1 \bar{x}_2 y$$

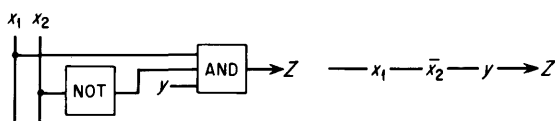


Figure 13-8

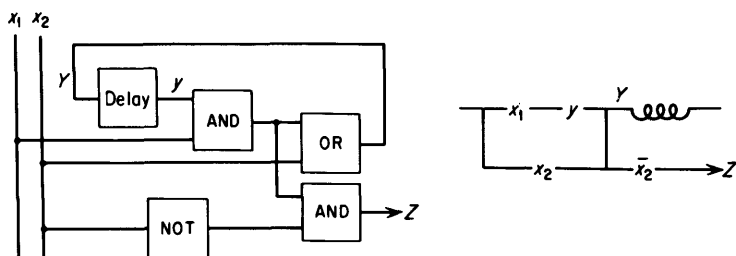


Figure 13-9

Sometimes a secondary operating sequence cannot be achieved unless more secondaries are used than at first appear necessary, as in Fig. 13-10. In this example, it is necessary to differentiate only between intervals 3 and 5, since these are the only two intervals with the same inputs but with different outputs.

Inspection of the timing chart shows that the differentiation could be achieved by having a secondary off during interval 3 and on during interval 5. This secondary, therefore, must turn on in interval 4. However, interval 4 is the same as interval 2, and if the secondary is turned on in interval 4, it would also turn on in interval 2 and be on in interval 3, and no differentiation would be accomplished. An attempt to have the secondary on during interval 3 and off during interval 5 would meet with the same result. Two secondaries are therefore required, and an operating sequence is shown in Fig. 13-11. y_1 turns on in interval 3, and stays on during interval 4, making interval 4 differ from interval 2. y_2 turns on in interval 4, and stays on during interval 5, making interval 5 differ from interval 3.

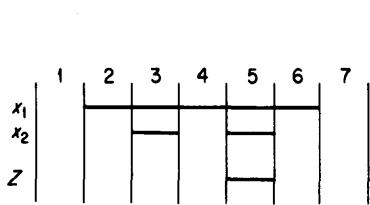


Figure 13-10

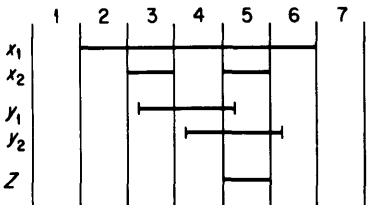


Figure 13-11

As sequential circuit requirements become more complex, the intuitive approach becomes less attractive; one may not be sure that the solution involves the minimum number of secondaries, or that the secondary operating sequence chosen is the one leading to the most economical circuit. For example, one secondary might turn on in any one of four intervals, and turn off in any one of three intervals, giving twelve possible operating sequences for that secondary alone. Sequential circuits are generally more complex when alternative sequences are possible.

EXAMPLE:

A sequential circuit is to function according to the timing chart in Fig. 13-12. Any of the four alternative sequences can occur. Only one secondary is required.

It is suggested that for an appreciation of the difficulty in solving such a problem intuitively, the reader solve this problem on his own before referring to the solution in Fig. 13-13.

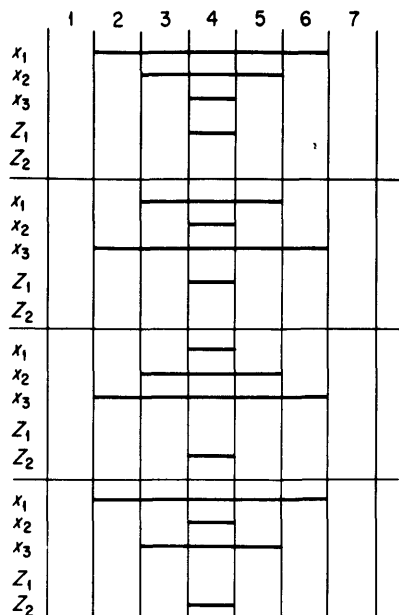
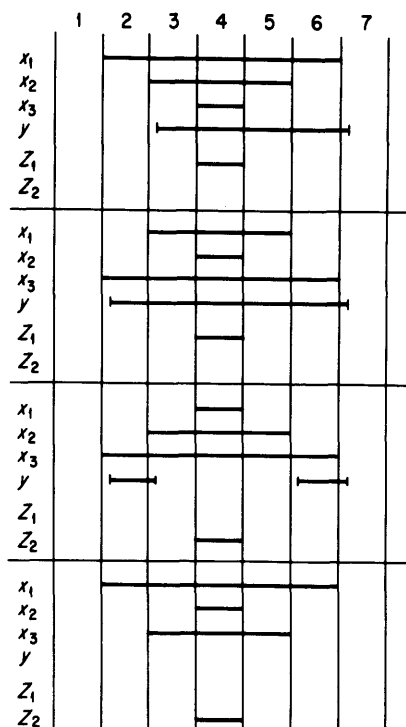


Figure 13-12



$$Y = x_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + x_1 y$$

$$Z_1 = x_1 x_2 x_3 y$$

$$Z_2 = x_1 x_2 x_3 \bar{y}$$

Figure 13-13

Flow Table

Preparatory to the examination of the formal method of sequential circuit design, the concept of the *flow table* will be discussed.

A flow table describes the circuit action of a sequential circuit. A flow table somewhat resembles a map, each entry being defined by a unique combination of variables. (In fact, in the synthesis procedure, secondary excitation or *Y*-maps, and output or *Z*-maps, are obtained from the flow tables.) The variables consist of all of the inputs (*x*'s) and secondaries (*y*'s); the inputs define the columns of the flow table, and the secondaries define the rows.

There is an entry in the table for every possible circuit state. Some of these states are stable, some are unstable, and some may be optional. For

a given input state, the stability or instability of a circuit state is solely a function of the secondaries. The concept of the flow table and its relationship to the secondary excitation map, or Y -map, is illustrated in Fig. 13-14. This example relates to the first example in the preceding section, and in particular to Figs. 13-15 and 13-16. Note the slight modification in the manner of drawing a map, the actual squares being omitted.

Referring first to the Y -map, the $x_1x_2y = 000$ entry represents a stable state, since the present state of the secondary, y , equals 0, and the excitation, or next state, of the secondary, Y , equals 0 (map entry for $x_1x_2y = 000$). The $x_1x_2y = 100$ entry also represents a stable state, since $y = Y = 0$. The $x_1x_2y = 111$ and $x_1x_2y = 101$ entries also represent stable states, since for these entries, $y = Y = 1$.

In the corresponding flow table, each stable state entry is denoted by an *arbitrary* circled number, ①, ②, ③, and ④, respectively, in the figure. All unstable state entries will be uncircled numbers, the number in each case denoting the stable state in which the circuit action will terminate.

The $x_1x_2y = 110$ entry represents an unstable state since $y = 0$ and $Y = 1$. Since $Y = 1$, the next state of the secondary will be 1, and the next circuit state will be $x_1x_2y = 111$. The $x_1x_2y = 110$ entry in the flow table is therefore an uncircled 3, denoting that secondary circuit action will terminate in stable state ③.

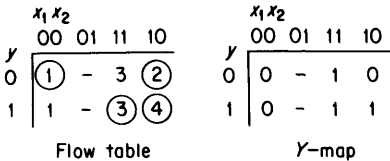


Figure 13-14

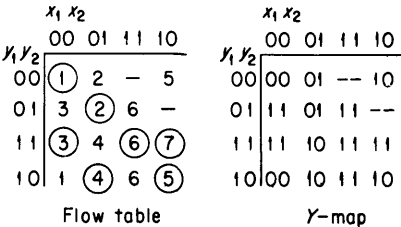


Figure 13-15

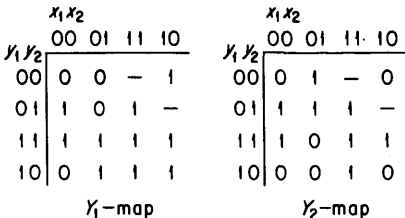


Figure 13-16

The $x_1x_2y = 001$ entry represents an unstable state, since $y = 1$ and $Y = 0$. Since $Y = 0$, the next state of the secondary will be 0, and the next circuit state will be 000. The $x_1x_2y = 001$ entry in the flow table is therefore an uncircled 1, denoting that secondary circuit action will terminate in stable state ①.

The $x_1x_2y = 010$ and $x_1x_2y = 011$ entries are optional, indicating either

that these circuit states can never occur, or that we do not care what the circuit action will be if they do occur.

Referring to the flow table and Y -map in Fig. 13-14, let us examine some possible circuit action. For example, assume that the circuit is initially in stable state ①, and that the inputs change from the $x_1x_2 = 00$ state to the $x_1x_2 = 10$ state. The circuit will now be in stable state ②, and no secondary action takes place. Now assume that the circuit is in stable state ②, and that the inputs change from $x_1x_2 = 10$ to $x_1x_2 = 11$. The circuit will now be in unstable state 3 (in this state, $y = 0$ and $Y = 1$), and after a delay, the secondary state will change from $y = 0$ to $y = 1$, circuit action terminating in stable state ③. Or assume that the circuit is in stable state ④, and that the inputs change from $x_1x_2 = 10$ to $x_1x_2 = 00$. The circuit will now be in unstable state 1 (in this state, $y = 1$ and $Y = 0$), and after a delay, the secondary will change from $y = 1$ to $y = 0$, circuit action terminating in stable state ①.

An example of a flow table with two secondaries is shown in Fig. 13-15, with the associated Y -map. The Y -map is actually a Y_1 -map and Y_2 -map superimposed, all left-hand entries defining Y_1 and all right-hand entries, Y_2 . If the maps were drawn separately, they would appear as in Fig. 13-16.

As an example of some possible circuit action related to the above flow table and Y -map, assume that the circuit is in stable state ⑥ and that the inputs change from $x_1x_2 = 11$ to $x_1x_2 = 01$. The circuit will now be in unstable state 4; in this state y_1 is stable ($y_1 = Y_1 = 1$), but y_2 is unstable ($y_2 = 1$ and $Y_2 = 0$). After a delay, y_2 will change from $y_2 = 1$ to $y_2 = 0$, and circuit action will terminate in stable state ④.

Note that a change in input states is represented by a horizontal movement in the flow table, while a change in secondary states is represented by a vertical movement.

Optional states may arise either because certain transitions can never occur, or because we don't care what the circuit action is for a particular transition.

14

Sequential Circuits II

Synthesis of Sequential Circuits

In the synthesis of sequential circuits, the circuit requirements are first completely described in a flow table. Systematic methods are then used to simplify and modify the flow table. The flow table is then transformed into maps that are read in the usual combinational sense to give the secondary excitation (Y) and output (Z) circuit expressions.

In more detail, the steps in the synthesis procedure are as follows:

- (1) A *primitive flow table* is constructed from the word statement of the problem.
- (2) The primitive flow table is tested for redundant states, and the number of stable states can be reduced if redundancy is found.
- (3) A *merged flow table* is obtained by *merging* rows of the primitive flow table. A *merger diagram* is used to obtain an optimum merger.
- (4) A *secondary state assignment* is made for the merged flow table. A *transition map* is used to determine the assignment.

- (5) A secondary excitation or *Y-map* is obtained from the flow table with secondary assignment, and the expressions for the secondary excitation circuits are read from the *Y-map*.
- (6) An output or *Z-map* is obtained from the flow table with secondary assignment and the primitive flow table: The output state for each stable state is identified in the primitive flow table, and its location in the *Z-map* is identified in the flow table with secondary assignment. Also, the actual state to state transitions are identified in the primitive flow table, this information being used in the assignment of output states for the unstable states. The output expressions are read from the *Z-map*.
- (7) The sequential circuit is drawn from the secondary excitation and output expressions. Circuit *hazards* must be checked for and eliminated.

Each of these steps in the procedure for the synthesis of sequential circuits will now be individually examined.

Primitive Flow Table

The first step in the synthesis of sequential circuits is the construction of a *primitive flow table* from the word statement of the problem. In a primitive flow table, each stable state (circled entry) is assigned a separate row. This implies that a different secondary state is assigned to each stable state, although actual secondary state assignments are not made at this time. It further implies that every input change is followed by a secondary change in accomplishing a transition from one stable state to another. These implications apply only to the *primitive* flow table, which is the initial step in the synthesis. The primitive flow table is later modified, and more than one stable state may be assigned the same row (secondary state), and transitions from one stable state to another may be accomplished by input changes only.

In the primitive flow table, the output state for each stable state is recorded at the right in the corresponding row.

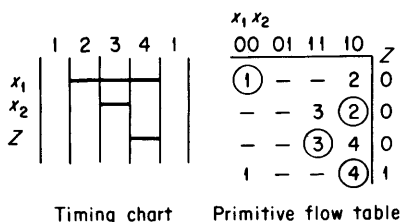


Figure 14-1

For study, the simple sequential circuit problem, solved intuitively in Chapter 13, is summarized in Fig. 14-1, together with a related primitive flow table.

The flow table stable state numbers are arbitrarily chosen to correspond with the timing chart interval numbers. Since the fifth interval

is equivalent to the first interval, it is assigned the same number. The optional entries in the flow table indicate circuit states that can never occur. The flow table can be seen to represent a complete description of the required circuit action.

EXAMPLE:

A sequential switching circuit is to have two inputs, x_1 and x_2 , and one output, Z . Z is to turn on when x_2 turns on, provided x_1 is already on. Z is to turn off when x_2 turns off. Only one input can change state at a time.

Development of the primitive flow table can be started by first considering the sequence for turning on the output (Fig. 14-2). All optional entries are due to the restriction to single changes of input.

If the circuit is in stable state ②, and the inputs change from $x_1x_2 = 10$ to $x_1x_2 = 00$, the circuit can return to stable state ①.

If the circuit is in stable state ③, and the inputs change from $x_1x_2 = 11$ to $x_1x_2 = 10$, the circuit can return to stable state ② (the output changing from $Z = 1$ to $Z = 0$). If the circuit is in stable state ③, and the inputs change from $x_1x_2 = 11$ to $x_1x_2 = 01$, the circuit must change to a new stable state ④, for which $Z = 1$.

If the circuit is in stable state ④, and the inputs change from $x_1x_2 = 01$ to $x_1x_2 = 11$, the circuit can return to stable state ③. If the circuit is in stable state ④, and the inputs change from $x_1x_2 = 01$ to $x_1x_2 = 00$, the circuit can return to stable state ① (the output changing from $Z = 1$ to $Z = 0$). The primitive flow table at this stage of development is shown in Fig. 14-3.

If the circuit is in stable state ①, and the inputs change from $x_1x_2 = 00$ to $x_1x_2 = 01$, the circuit must change to a new stable state ⑤, for which $Z = 0$.

If the circuit is in stable state ⑤, and the inputs change from $x_1x_2 = 01$ to $x_1x_2 = 00$, the circuit can return to stable state ①. If the circuit is in stable state ⑤, and the inputs change from $x_1x_2 = 01$ to $x_1x_2 = 11$, the circuit must change to a new stable state ⑥, for which $Z = 0$.

x_1x_2						Z
00	01	11	10			
①		—	2			0
	—	3	②			0
—		③				1

Figure 14-2

x_1, x_2						Z
00	01	11	10			
①		—	2			0
1	—	3	②			0
—	4	③	2			1
1	④	3	—			1

Figure 14-3

$x_1 \ x_2$						Z
00	01	11	10			
①	5	—	2			0
1	—	3	②			0
—	4	③	2			1
1	④	3	—			1
1	⑤	6	—			0
—	5	⑥	2			0

Figure 14-4

If the circuit is in stable state ⑥, and the inputs change from $x_1x_2 = 11$ to $x_1x_2 = 01$, the circuit can return to stable state ⑤. If the circuit is in stable state ⑥, and the inputs change from $x_1x_2 = 11$ to $x_1x_2 = 10$, the circuit can return to stable state ②. The completed primitive flow table is shown in Fig. 14-4.

Construction of the primitive flow table forces the logical designer to completely account for all possible circuit action. There may have been certain input sequences that the designer had not initially considered. However, in the construction of the flow table, these sequences are called to his attention, and he must decide what the circuit action will be when these sequences occur (or else determine that the circuit action is optional).

"Power-on" Output State

The output specifications of a sequential circuit may be complete with regard to the circuit action required once the circuit is "in operation," but the output state when the power is first turned on may be arbitrary. For example, consider the following circuit requirement:

A sequential circuit has two inputs, x_1 and x_2 , and two outputs, Z_1 and Z_2 . Only one input can change at a time, and the input state $x_1x_2 = 11$ can never occur. When

$$x_1x_2 = 01 \quad \text{or} \quad 10, \quad Z_1Z_2 = 00$$

When

$$x_1x_2 = 00, \quad \text{following} \quad x_1x_2 = 01, \quad Z_1Z_2 = 01$$

When

$$x_1x_2 = 00, \quad \text{following} \quad x_1x_2 = 10, \quad Z_1Z_2 = 10$$

If $x_1x_2 = 00$ when the power is first turned on, what should the output be?

It may be that either output $Z_1Z_2 = 01$ or $Z_1Z_2 = 10$ may be arbitrarily chosen as the "power-on" output state. If so, assuming that stable state ① is the power-on stable state, either of the two primitive flow tables in Fig. 14-5 would be satisfactory.

x_1x_2				Z_1Z_2
00	01	11	10	
① 3 - 4	01			
② 3 - 4	10			
1 ③ - -	00			
2 - - ④	00			

x_1x_2				Z_1Z_2
00	01	11	10	
① 3 - 4	10			
② 3 - 4	01			
2 ③ - -	00			
1 - - ④	00			

Figure 14-5

x_1x_2				Z_1Z_2
00	01	11	10	
① 4 - 5	00			
② 4 - 5	01			
③ 4 - 5	10			
2 ④ - -	00			
3 - - ⑤	00			

Figure 14-6

If one of the two output conditions, $Z_1Z_2 = 01$ or $Z_1Z_2 = 10$, is preferred and specified as the power-on output state, only one of the above flow tables would, of course, be satisfactory.

Still another possibility is that $Z_1Z_2 = 00$ may be desired as the power-on output state. If so, the primitive flow table in Fig. 14-6 would describe the circuit action. Note that once circuit action starts, causing a movement out of the first row of the flow table, the circuit will never return to stable state ①. Stable state ① thus serves only as a power-on stable state. Unless they are specifically required, such additional stable states should be avoided, since they generally lead to less economical circuits.

Elimination of Redundant Stable States

In the construction of the primitive flow table, it is possible to introduce more stable states than are needed. This is sometimes done inadvertently because it is not apparent that two or more stable states are actually equivalent. If two stable states are equivalent, one of them is redundant and may be removed, eliminating a row of the primitive flow table. Once the primitive flow table has been completed, then, the next step in the synthesis is to test for any redundant stable states that may be present.

Two stable states are equivalent if:

- (1) They have the same input state (they are in the same column),
- and (2) They have the same output state,
- and (3) For each possible input change there is a transition from these stable states to the same or equivalent states.

EXAMPLE 1:

In the primitive flow table of Fig. 14-7, stable states ② and ④ are equivalent. They have the same input state, $x_1x_2 = 01$; they have the same output state, $Z_1Z_2 = 10$; and since the remaining entries in both the second and fourth rows are identical, column for column, for each possible

x_1x_2				Z_1Z_2
00	01	11	10	
①	2	—	6	00
1	②	5	—	10
③	4	—	6	10
1	④	5	—	10
—	4	⑤	6	01
3	—	5	⑥	11

Figure 14-7

$x_1 x_2$				$Z_1 Z_2$
00	01	11	10	
①	2	—	6	00
1	②	5	—	10
③	2	—	6	10
—	2	⑤	6	01
3	—	5	⑥	11

Figure 14-8

input change there is a transition from these stable states to the same state.

When two stable states are equivalent, it is customary to eliminate the one with the higher number. All occurrences of the higher number are replaced by the lower number, and the row containing the higher-numbered stable state is eliminated entirely. The primitive flow table with the redundant stable state removed is shown in Fig. 14-8.

EXAMPLE 2:

In the primitive flow table of Fig. 14-9, the equivalence of stable states ② and ④ can be immediately established as in Example 1. Stable states ① and ③ have the same input state and the same output state, and therefore the first two criteria for equivalence are satisfied. Examination of the first and third rows shows that the remaining entries are identical, column for column, except for the $x_1x_2 = 01$ column. In this column, there is a 2 in the first row, and a 4 in the third row. The equivalence of stable states ① and ③, therefore, is dependent upon the equivalence of stable states ② and ④. Since the equivalence of ② and ④ has been established, ① and ③ are equivalent also. The reduced primitive flow table is shown in Fig. 14-10.

$x_1 x_2$				$Z_1 Z_2$
00	01	11	10	
① 2	—	6	00	00
1 ②	5	—	10	10
③ 4	—	6	00	00
1 ④	5	—	10	10
—	7	⑤	6	01
3	—	5	⑥	11
3	⑦	5	—	11

Figure 14-9

x_1x_2				Z_1Z_2
00	01	11	10	
① 2	—	6	00	
1 ②	5	—	10	
—	7	⑤	6 01	
1	—	5	⑥ 11	
1 ⑦	5	—	11	

Figure 14-10

x_1x_2				Z_1Z_2
00	01	11	10	
① 2	—	6	00	00
3 ②	5	—	10	10
③ 4	—	6	00	00
1 ④	5	—	10	10
—	7 ⑤	6	01	01
3	—	5 ⑥	11	11
3 ⑦	5	—	11	11

Figure 14-11

EXAMPLE 3:

In the primitive flow table of Fig. 14-11, the equivalence of stable states ① and ③ is dependent upon the equivalence of stable states ② and ④. The equivalence of ② and ④, however, is dependent upon the equivalence of ① and ③. If ① and ③ are made equivalent and ② and ④ are made equivalent, analysis will show that the circuit action is the same as that prescribed by the original flow table. The reduced primitive flow table for this example is identical to that in Example 2.

Equivalences can thus be made when they are interdependent upon each other, or when they are dependent upon other established equivalences. The requirements for equivalence can, in fact, be stated in another, and

perhaps more directly usable, way: *two stable states with the same input state and the same output state can be made equivalent unless the equivalence depends upon a nonequivalence.*

It follows that an efficient approach in testing a primitive flow table for redundant stable states is to establish all *nonequivalences* (within a column) first; all pairs of stable states, with the same input state and the same output state, not established as nonequivalent can then be made equivalent. Example 4 illustrates this approach.

EXAMPLE 4:

Examination of the primitive flow table of Fig. 14-12 shows that there are possible equivalences between stable states ①, ④, and ⑫; between ②, ⑦, and ⑨; between ⑤, ⑧, and ⑪; and between ③ and ⑩. Immediately established nonequivalences (within a column) are between stable states ③ and ⑩, and between ⑧ and ⑩.

$x_1 x_2$				$Z_1 Z_2$
00	01	11	10	
①	2	6	3	00
4	②	5	3	11
1	7	6	③	01
④	7	5	8	00
1	9	⑤	3	10
12	2	⑥	10	10
1	⑦	11	3	11
12	9	11	⑧	01
12	⑨	6	8	11
1	7	6	⑩	11
1	2	⑪	10	10
⑫	2	11	8	00

Figure 14-12

To aid in establishing further nonequivalences, a tabular approach can be helpful: A table is constructed with a row for each possible equivalence, and a column for each possible equivalence and established nonequivalence. All nonequivalences are circled for identification. Check marks are placed in the proper locations of the table, a check mark indicating that the possible equivalence in the corresponding row is dependent upon the possible equivalence or established nonequivalence in the corresponding column.

The table for Example 4 is shown in Fig. 14-13.

The nonequivalences 3-10 and 8-10 are circled for identification. Any

stable states whose equivalence is dependent upon a nonequivalence must themselves be nonequivalent. In Fig. 14-13, the check marks in the 3-10 column establish that the stable states ⑤ and ⑧ are nonequivalent, and that the stable states ⑤ and ⑪ are nonequivalent. The 5-6 and 5-11 column

	①-④	1-12	④-12	②-7	②-9	7-9	⑤-6	⑤-11	6-11	3-8	③-10	⑧-10
1-4				✓			✓			✓		
1-12									✓	✓		
4-12				✓				✓				
2-7	✓							✓				
2-9			✓				✓			✓		
7-9		✓							✓	✓		
5-6		✓			✓						✓	
5-11					✓						✓	
6-11		✓										
3-8		✓				✓			✓			

Figure 14-13

$x_1 x_2$	00	01	11	10	$Z_1 Z_2$
①	2	6	3	00	
4	②	5	3	11	
1	7	6	③	01	
④	7	5	3	00	
1	7	⑤	3	10	
1	2	⑥	10	10	
1	⑦	6	3	11	
1	7	6	⑩	11	

Figure 14-14

designation are therefore circled. The check marks in these two columns establish that 1-4, 2-9, 4-12, and 2-7 are nonequivalences. These column designations are also circled, and the check marks in these four columns indicate the nonequivalence of 2-7, 2-9, 1-4, 4-12, 5-6, and 5-11. These nonequivalences have already been established, and since no new nonequivalences are found, the procedure is completed. All uncircled column designations indicate equivalences that can be made: 1-12, 7-9, 6-11, and 3-8.

The primitive flow table can therefore be reduced to eight rows (Fig. 14-14).

Pseudo-Equivalence

Two stable states may be equivalent in all respects except for one or both of the following conditions:

- (1) For a given input change, there is a transition from one of these stable states to a prescribed state, whereas the transition from the second stable state is optional.
- (2) An output state associated with one of these stable states is prescribed, whereas for the second stable state, the corresponding output state is optional.

If either of the above conditions exists, the two stable states are said to be *pseudo-equivalent*, and can be considered as equivalent. These conditions are illustrated by the following two examples.

Optional Transition

x_1x_2					Z_1Z_2
00	01	11	10		
①	3	5	4	00	≡
②	3	—	4	00	

x_1x_2					Z_1Z_2
00	01	11	10		
①	3	5	4	00	≡
②	3	5	4	00	

Figure 14-15

Stable states ① and ② are pseudo-equivalent, since an input change from $x_1x_2 = 00$ to $x_1x_2 = 11$ results in a transition from stable state ① to ⑤, whereas the transition from stable state ② is optional. Since the optional entry can be replaced with a 5, stable states ① and ② can be made equivalent.

Optional Output

x_1x_2					Z_1Z_2
00	01	11	10		
①	3	5	4	00	≡
②	3	5	4	0—	

x_1x_2					Z_1Z_2
00	01	11	10		
①	3	5	4	00	≡
②	3	5	4	00	

Figure 14-16

In this example, stable states ① and ② are pseudo-equivalent, since for stable state ①, $Z_2 = 0$, whereas for stable state ②, Z_2 is optional. Since the optional entry can be replaced with a 0, stable states ① and ② can be made equivalent.

A stable state may be pseudo-equivalent to two or more other stable states which themselves are nonequivalent.

EXAMPLE:

x_1x_2				Z
00	01	11	10	
①	4	—	7	0
②	5	6	—	0
③	—	6	7	0
1	④	6	—	1
2	⑤	6	—	0

Figure 14-17

Stable states ① and ③ are pseudo-equivalent, stable states ② and ③ are pseudo-equivalent, whereas stable states ① and ② are nonequivalent. The three possible reductions are shown in Fig. 14-18.

$x_1 x_2$				Z
00	01	11	10	
① 4	6	7	0	0
② 5	6	—	0	0
1 ④	6	—	1	1
2 ⑤	6	—	0	0

① \equiv ③

$x_1 x_2$				Z
00	01	11	10	
① 4	—	7	0	0
② 5	6	7	0	0
1 ④	6	—	1	1
2 ⑤	6	—	0	0

② \equiv ③

$x_1 x_2$				Z
00	01	11	10	
① 4	6	7	0	0
② 5	6	7	0	0
1 ④	6	—	1	1
2	6	—	0	0

① \equiv ③
② \equiv ③

Figure 14-18

In the last reduction, stable state ③ can be made equivalent to both ① and ② since the optional entry in the $x_1 x_2 = 01$ column can be replaced with a 4 or 5.

Although not the case in the example above, maximum state reduction may require that a stable state be made equivalent to two or more other stable states which themselves are nonequivalent.

EXAMPLE:

$x_1 x_2$				
00	01	11	10	
①	—	—	—	0
1	②	—	5	0
1	③	—	6	1
1	④	—	7	—
1	3	—	⑤	1
1	2	—	⑥	1
1	4	—	⑦	1

Figure 14-19

$x_1 x_2$						
00	01	11	10			Z
① 2 or 3	—	5 or 6	0			0
1 ②	—	5	0			0
1 ③	—	6	1			1
1 3	—	⑤	1			1
1 2	—	⑥	1			1

Figure 14-20

Stable states ② and ③ are nonequivalent, and therefore stable states ⑤ and ⑥ are also nonequivalent. Stable states ② and ④ can be made equivalent if stable states ⑤ and ⑦ can be made equivalent; the equivalence of ⑤ and ⑦, in turn, is dependent upon the equivalence of ③ and ④; the equivalence of ③ and ④ is dependent upon the equivalence of ⑥ and ⑦; and the equivalence of ⑥ and ⑦ is dependent upon the equivalence of ② and ④. The following equivalences can therefore be made

$$\textcircled{2} \equiv \textcircled{4}$$
$$\textcircled{3} \equiv \textcircled{4}$$

$$\textcircled{5} \equiv \textcircled{7}$$
$$\textcircled{6} \equiv \textcircled{7}$$

and the reduced table appears as in Fig. 14-20. Note that the 4 and 7 in the first row can each be replaced with either of two equivalences.

A flow table with pseudo-equivalences is also called an incompletely specified flow table. In general, reduction of such a flow table to the minimum number of rows can involve trial and error procedures (see Related Literature section).

PROBLEMS

1. A sequential circuit is to have two inputs, x_1 and x_2 , and one output, Z . The inputs represent, in binary, the numbers 0 through 3.

x_1x_2	Number representation
00	0
01	1
10	2
11	3

- If a change in input increases the represented number by *one*, the output is to turn on, if not already on. If a change in input decreases the represented number by *one*, the output is to change state. No other input change is to cause any change in output. All input changes are possible. Draw a primitive flow table for this circuit requirement.
2. A sequential circuit is to have two inputs, x_1 and x_2 , and one output, Z . The inputs represent, in binary, the numbers 0 through 3. If a change in input increases the represented number, the output is to turn on, if not already on. If a change in input decreases the represented number, the output is to turn off, if not already off. All input changes are possible except that both inputs will never turn off simultaneously. Draw a primitive flow table for this circuit requirement.
- *3. A sequential circuit is to have two inputs, x_1 and x_2 , and one output, Z . If the number of inputs that are *on* increases, the output is to turn off, if not already off. If the numbers of inputs that are *on* decreases, the output is to turn on, if not already on. No other input change is to cause any change in output. Both inputs will never turn off simultaneously; otherwise, all input changes are possible. Draw a primitive flow table for this circuit requirement.

4. Draw a primitive flow table equivalent to Fig. 14-21, but with no redundant stable states.

x_1x_2					Z
00	01	11	10		
①	6	9	11	0	0
②	6	7	10	0	0
③	4	8	10	0	0
3	④	9	11	1	1
1	⑤	9	12	1	1
1	⑥	8	10	1	1
1	5	⑦	11	0	0
3	6	⑧	11	0	0
1	4	⑨	10	0	0
3	6	9	⑩	1	1
1	4	8	⑪	1	1
2	4	8	⑫	0	0

Figure 14-21

- *5. Make all possible equivalences (Fig. 14-22) and draw a primitive flow table with a minimum of stable states.

x_1x_2					Z
00	01	11	10		
①	5	9	11	0	0
②	5	7	11	0	0
③	4	8	-	-	-
-	④	9	11	1	1
1	⑤	8	11	-	-
3	⑥	7	10	1	1
1	6	⑦	11	0	0
-	5	⑧	11	-	-
1	4	⑨	11	0	0
1	4	8	⑩	0	0
2	5	9	⑪	1	1

Figure 14-22

6. Test the primitive flow table of Fig. 14-23 for any redundant stable states that may be present.

$x_1 x_2$						$z_1 z_2$
00	01	11	10			
①	—	3	5			00
②	—	4	7			00
1	—	③	7			11
2	—	④	6			11
1	8	—	⑤			10
—	9	4	⑥			10
2	—	3	⑦			10
1	⑧	3	—			01
2	⑨	4	—			10

Figure 14-23

15

Sequential Circuits III

Merged Flow Table; Merger Diagram

After testing for and eliminating any redundant stable states, the next step is to *merge* rows of the primitive flow table and obtain a *merged flow table*. In the primitive flow table, each stable state is assigned a separate row, and all transitions between stable states involve both an input change and a secondary change. Merging reduces the number of rows in the flow table by placing more than one stable state in the same row. Transitions between stable states in the same row are realized by input changes only.

The advantage of merging is that by reducing the number of rows in the flow table, the number of required secondary states is reduced, and often, as a consequence, the number of required secondaries is reduced also. These reductions generally lead to greater circuit economy. It should be noted that merging reduces the number of rows of the flow table, but it does *not* reduce the number of stable states.

The rules for merging are as follows:

- (1) Two or more rows can merge if, within the rows, there are no conflicting state numbers in any column. For example, two rows can merge if each column contains either two like state numbers, one state number and a —, or two —'s.
- (2) All state numbers in the merging rows are written in the respective columns of the merged row. If a state number is circled in one of the merging rows, it is circled in the merged row, retaining the stable state designations.

The output states for each stable state, recorded in the primitive flow table, in no way affect merging, and are not repeated in the merged flow table. The primitive flow table is referred to later for this output information.

EXAMPLE:

The merger of the two rows in Fig. 15-1 is shown in Fig. 15-2.

$x_1x_2x_3$								Z
000	001	011	010	110	111	101	100	
①	2	—	3	—	4	—	5	0
—	②	6	—	—	4	7	—	1

Figure 15-1

$x_1x_2x_3$							
000	001	011	010	110	111	101	100
①	②	6	3	—	4	7	5

Figure 15-2

Generally, there is more than one way of merging the rows of a flow table, and the choice of mergers can affect circuit economy. In obtaining an optimum merger, a *merger diagram* is useful.

To construct a merger diagram, the stable state numbers are arranged in a basically circular array. The numbers are used here only to identify the rows of the primitive flow table. If, in the flow table, two rows can be merged, the corresponding stable state numbers in the merger diagram are connected by a line. All pairs of rows are examined for a possible merger, and after all connecting lines have been drawn, the merger diagram is inspected for the optimum way of merging. The aim, in general, is to merge so as to obtain the minimum number of rows in the merged flow table.

A primitive flow table and its associated merger diagram are shown in Fig. 15-3. Referring to the merger diagram, note that row 2 can merge with row 1 or row 3, but that rows 1 and 3 cannot merge with each other. Therefore, rows 1, 2, and 3 cannot all merge into one row, and a choice must be made between the merger of rows 1 and 2 and the merger of rows 2 and 3.

A merger of rows 1, 2, and 5 cannot be made for the same reason, and a

$x_1 x_2$		00	01	11	10	Z
①	5	—	2			0
1	—	3	②			1
—	6	③	2			0
—	6	④	2			1
1	⑤	4	2			1
1	⑥	4	—			0

Primitive flow table

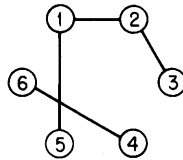


Figure 15-3

$x_1 x_2$		00	01	11	10	
①	⑤	4	2			
1	6	③	②			
1	⑥	④	2			

Merged flow table

Figure 15-4

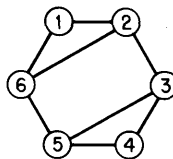
choice must be made between the merger of rows 1 and 2 and the merger of rows 1 and 5.

The mergers of rows 1 and 5, 2 and 3, and 4 and 6 result in a three-row flow table, which is optimum. A merger of rows 1 and 2 would not be desirable, since it would leave rows 3 and 5 unmerged, and the resulting flow table would contain four rows. The optimum three-row merged flow table is shown in Fig. 15-4.

Figure 15-5 illustrates three-row mergers. Rows 1, 2 and 6 can all merge into one row, as can rows 3, 4 and 5. Note that a four-row merger between rows 2, 3, 5 and 6 is not possible, since rows 2 and 5 cannot merge and rows 3 and 6 cannot merge.

$x_1 x_2$		00	01	11	10	Z
①	5	6	2			0
1	—	—	②			1
—	5	③	2			0
④	5	3	—			0
4	⑤	—	—			1
—	5	⑥	2			1

Primitive flow table



Merger diagram

$x_1 x_2$		00	01	11	10	
①	5	⑥	②			
④	⑤	③	2			

Merged flow table

Figure 15-5

Figure 15-6 is an example of a four-row merger. Rows 1, 2, 5, and 6 can all merge into one row, and there is also a two-row merger between rows 3 and 4.

Often there may be more than one way of obtaining a minimum-row merger. In Fig. 15-7, there are four different ways of reducing to a four-row merged flow table. When there is more than one minimum-row merger, all of them should be considered, since there is no way of knowing at this stage of design which merger will result in the most economical circuit.

Once the merged flow table has been obtained, the next step is the

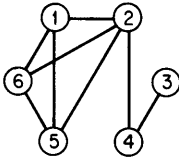


Figure 15-6

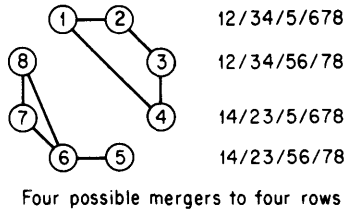


Figure 15-7

assignment of secondary states to the rows of the flow table. Following this, a secondary excitation or *Y-map* is obtained from the flow table with secondary assignment. The expressions for the secondary excitation circuits are read from the *Y-map*. Before these steps are examined, however, the concepts of *cycles*, *noncritical races*, and *critical races* should be understood.

Cycles

Until now, the discussion of unstable states has been limited to the case in which following a secondary change a stable state is reached. The case will now be considered in which an unstable state leads to another unstable state. Such a succession of two or more secondary changes is called a *cycle*. An example of a cycle is illustrated in the $x_1x_2 = 00$ column of the flow table and associated *Y-map* in Fig. 15-8.

In a flow table (Fig. 15-8) all unstable state numbers correspond to the stable state that will be reached when all secondary circuit action terminates. Arrows are used to indicate the movement from an unstable state to another unstable state. The absence of an arrow leading from an unstable state number indicates that the next state is the corresponding stable state.

In this flow table and associated *Y-map*, if the circuit is in stable state ②, and there is an input change from $x_1x_2 = 01$ to 00, there will be a cycle of three successive secondary changes before stable state ① is reached: $y_1y_2 = 10$ to 11 to 01 to 00.

If the circuit is in stable state ③, and there is an input change from $x_1x_2 = 10$ to 00, there will be a cycle of only two successive secondary changes before stability is reached: $y_1y_2 = 11$ to 01 to 00.

If the circuit is in stable state ④, and there is an input change from

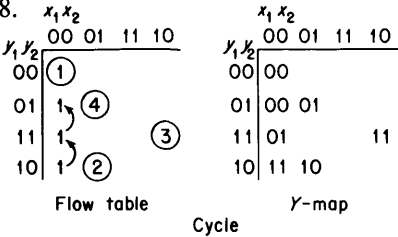


Figure 15-8

$x_1x_2 = 01$ to 00 , stable state ① is reached following the single secondary change from $y_1y_2 = 01$ to 00 .

Races

In the cycles and single secondary changes discussed so far, each secondary excitation differed from the present secondary state in only one variable, that is, only one secondary was unstable at a time. If more than one secondary is unstable at a time, a *race* condition is said to exist.

An example of a race is illustrated in the $x_1x_2 = 00$ column of the flow table and associated Y -map in Fig. 15-9.

x_1x_2			x_1x_2		
y_1	y_2		y_1	y_2	
00	00	①	00	00	
01	01	1	01	00	
11	11	1 ②	11	00 11	
10	10	1	10	00	

Flow table Y -map

Non-critical race

Figure 15-9

If the circuit is in stable state ②, and there is an input change from $x_1x_2 = 01$ to 00 , the excitation, $Y_1Y_2 = 00$ will differ from the secondary state, $y_1y_2 = 11$, in two variables, and both secondaries will attempt to change state

at the same time. However, the physical response times of the secondaries may differ, and one secondary may respond faster than the other. If both secondaries respond at the same time, the next secondary state will be $y_1y_2 = 00$, and no further secondary action will take place since this state is stable. If y_1 responds first, the next secondary state will be $y_1y_2 = 01$. This state is unstable, and a further secondary change to the stable $y_1y_2 = 00$ state will take place. If y_2 responds first, the next secondary state will be $y_1y_2 = 10$. This state is unstable, and a further secondary change to the stable $y_1y_2 = 00$ state will take place.

In the example above, no matter what the outcome of the race (y_1 responds first, y_2 responds first, or y_1 and y_2 respond together), circuit action terminates in the desired stable state. Such a race is termed *noncritical*.

A more complex example is shown in Fig. 15-10. If the circuit is in stable state ②, and there is an input change from $x_1x_2 = 01$ to 00 , there will be a race condition from the $y_1y_2y_3 = 111$ state to the $y_1y_2y_3 = 010$ state. Depending upon the outcome of the race, the next secondary state may be $y_1y_2y_3 = 010$, 110 , or 011 . The 010 state changes to the stable 000 state. The 110 state cycles through the 100 state to the stable 000 state.

The 011 state attempts to change to the 000 state: another race condition, the next secondary state being 000 (stable), 010 or 001 . The 010 and 001 states both change to the stable 000 state. Therefore, no matter what the outcome of the races, the circuit action eventually terminates in stable state $y_1y_2y_3 = 000$, and the races are thus noncritical. All possible circuit

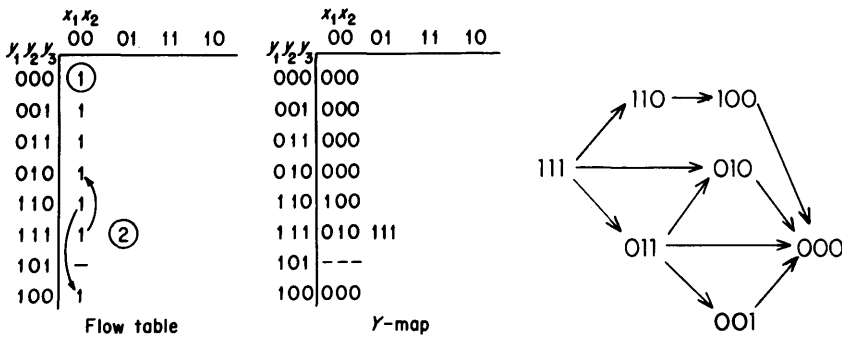


Figure 15-10

actions are illustrated diagrammatically at the right in Fig. 15-10.

If a race can terminate in any of two or more nonequivalent stable states (or can endlessly cycle), it is said to be a *critical race*. An example of a critical race is illustrated in the $x_1 x_2 = 00$ column of the flow table and associated Y-map in Fig. 15-11.

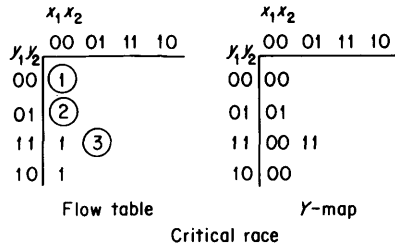


Figure 15-11

If the circuit is in stable state ③, and there is an input change from $x_1 x_2 = 01$ to 00 , there will be a race condition from the $y_1 y_2 = 11$ state to the $y_1 y_2 = 00$ state. Depending upon the outcome of the race, the next secondary state may be $y_1 y_2 = 00, 01$, or 10 . If both secondaries respond at the same time, the next secondary state will be $y_1 y_2 = 00$ (stable state ①). If y_2 responds first, the next secondary state will be $y_1 y_2 = 10$, followed by a further secondary change to $y_1 y_2 = 00$ (stable state ①). If y_1 responds first, the next secondary state will be $y_1 y_2 = 01$ (stable state ②), which is not the circuit action desired.

The behavior of a circuit with a critical race condition is thus not predictable, and critical races therefore represent improper design and must be avoided.¹

However, noncritical races and cycles are not only permissible in sequential circuit design, but they may be desirable.

Cycles may be used to avoid critical races or to introduce desired additional time delays in secondary transitions. Noncritical races are useful where short transition times are desirable. Cycles or noncritical races may lead to the most economical circuit in a given case.

¹Critical races may be avoided if one secondary, inherently or by the insertion of additional delay, responds slower than another. In the discussion that follows, however, it will be assumed that the relative response times are indeterminate.

Secondary State Assignment and Y-Map

The next two steps in the synthesis of sequential switching circuits are (a) making secondary state assignments to the rows of the merged flow table and (b) obtaining a Y -map from the flow table with secondary assignment and reading the secondary excitation circuit expressions from the Y -map. It is advantageous to consider these two steps concurrently.

Arbitrarily, stable state ① will always be placed in the first row of the flow table, and will be assigned the all-0 state: all input states equal 0 and all secondary states equal 0. The all-0 secondary state will thus always be assigned to the first row.

A two-row flow table presents no secondary assignment problems. One secondary is required, with the assignment $y = 0$ for the first row, and $y = 1$ for the second row. In making secondary assignments to flow tables of three or more rows, we cannot assign secondary states arbitrarily, or critical races may result. For these flow tables, a transition map is helpful in deriving assignments with the minimum number of variables, and that are free of critical races.

Transition Map

The secondaries are the variables of a transition map, each square in the map representing a secondary state. With two secondaries, four secondary states are possible; with three secondaries, eight secondary states are possible, and so forth. Two- and three-variable transition maps are shown in Fig. 15-12.

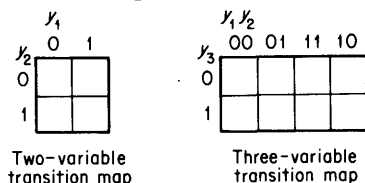


Figure 15-12

As a first step in determining a secondary assignment, each row in the merged flow table is assigned a letter reference. The assignment of a secondary state to a row in the flow table is associated

with the entry of the row reference letter in the corresponding square of the transition map.

In the assignment of secondary states to the rows of a merged flow table, the possible row-to-row transitions must be examined: if there is a transition between two particular rows, the secondary states for the two rows must either differ in only one variable, or if they differ in more than one, either a cycle or a noncritical race must be prescribed. *Critical races must be avoided.*

Note that secondary state assignments differing in only one variable are

represented in the transition map by “adjacent” entries. Therefore, if there is a transition between two particular rows, their reference letters must either appear in adjacent squares of the map, or if not, either a cycle or a noncritical race must be prescribed.

EXAMPLE:

Row *a* (Fig. 15-13) is arbitrarily assigned the $y_1y_2 = 00$ state, and an *a* is entered in the corresponding square of the transition map.

Examination of the flow table shows that all row-to-row transitions must be accomplished by single secondary changes, since no cycles or noncritical races are possible.

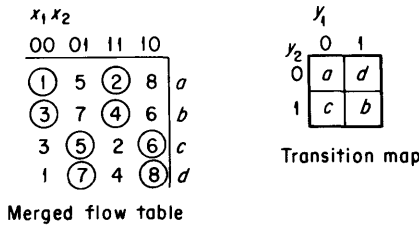


Figure 15-13

There is a transition between rows *a* and *c* (stable states ① or ② to ⑤). Therefore, the secondary assignment for row *c* must differ from the row *a* assignment in only one variable, that is, the assignment for row *c* must be either $y_1y_2 = 01$ or 10 . The $y_1y_2 = 01$ assignment is arbitrarily chosen, a *c* being entered in the corresponding square of the transition map.

There is also a transition between rows *a* and *d* (stable states ① or ② to ⑧). Row *d* must therefore be assigned the $y_1y_2 = 10$ state, a *d* being entered in the corresponding square of the transition map.

The transition between rows *b* and *d* (stable states ③ or ④ to ⑦) requires that the two corresponding row assignments differ in only one variable, and row *b* must be assigned the remaining $y_1y_2 = 11$ state, a *b* being entered in the corresponding square of the transition map.

It remains to be determined whether this assignment satisfies all other transitions in the flow table. The remaining transitions are between rows

- b* and *c* (stable states ③ or ④ to ⑥)
- c* and *b* (stable states ⑤ or ⑥ to ③)
- c* and *a* (stable states ⑤ or ⑥ to ②)
- d* and *a* (stable states ⑦ or ⑧ to ①)
- d* and *b* (stable states ⑦ or ⑧ to ④)

All transitions are between adjacent map entries, and the secondary

		$x_1 x_2$				
		00	01	11	10	
$y_1 y_2$	00	①	5	②	8	a
	01	3	⑤	2	⑥	c
	11	③	7	④	6	b
	10	1	⑦	4	⑧	d

Flow table with secondary assignment

Figure 15-14

assignment, as indicated in the transition map, is therefore satisfactory. The flow table with secondary assignment is shown in Fig. 15-14.

Note that the order of the rows is rearranged so that the secondary states are in the reflected ordering, preparatory to obtaining excitation and output maps from the flow table.

An assignment free of critical races is not always possible with the initial number of variables, and additional secondaries may be required, as will be described in the following chapter.

Y-Map

In obtaining a Y -map from the flow table with secondary assignment, it is convenient to make the map entries for the stable states first. Each map entry (secondary excitation) corresponding to a stable state will be the same as the present secondary state. A partially completed Y -map from the preceding flow table is shown in Fig. 15-15.

Each map entry (secondary excitation) corresponding to an unstable state will be the same as the next secondary state. Since no cycles are prescribed in this example, each map entry corresponding to an unstable state will be the same as the map entry for the corresponding stable state. The completed Y -map is shown in Fig. 15-16.

All left-hand map entries in Fig. 15-16 define Y_1 , and all right-hand entries define Y_2 . The expressions for the secondary excitation circuits follow.

$$Y_1 = \bar{x}_1 \bar{x}_2 y_2 + x_1 \bar{x}_2 \bar{y}_2 + x_2 y_1$$

$$Y_2 = \bar{x}_1 x_2 \bar{y}_1 + x_1 x_2 y_1 + \bar{x}_2 y_2$$

When, in a column of a flow table, an unstable state number appears more than once, a cycle or a noncritical race may be prescribed in that column.

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00	00		00	
	01		01		01
	11	11		11	
	10		10		10

Figure 15-15

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00	00	01	00	10
	01	11	01	00	01
	11	11	10	11	01
	10	00	10	11	10

Y -map

Figure 15-16

EXAMPLE:

$x_1 x_2$		00	01	11	10	
0	1	①	2	⑦	4	a
1	0		⑤	③	6	b
1	1		②	3	⑥	c
1	0		5	7	④	d

Merged flow table

Figure 15-17

x_1		0	1
y_2	0	a	d
1	c	b	

Transition map

$x_1 x_2$		00	01	11	10	
$y_1 y_2$	00	①	2	⑦	4	a
01			②	3	⑥	c
11			⑤	③	6	b
10			5	7	④	d

Partially-completed flow table with secondary assignment

Figure 15-18

A column having only one stable state need not be of any concern in making a secondary assignment, since cycles and noncritical races can be prescribed in such a column. Therefore, ignoring the $x_1 x_2 = 00$ column temporarily (Fig. 15-17), the row-to-row transitions in the other three columns lead to the secondary assignment shown in Fig. 15-18.

The optional ways that the $x_1 x_2 = 00$ column can be treated are shown in Figs. 15-19 and 15-20. It should be remembered that transitions from one unstable state to another are denoted by arrows; the absence of an arrow leading from an unstable state number indicates that the next state is the corresponding stable state. The options are numbered for reference.

		#1	#2	#3	#4	#5	#6	#7
$x_1 x_2$	$y_1 y_2$	00	00	00	00	00	00	00
00		①	①	①	①	①	①	①
01		1	1	1	1	1	1	1
11		1	1	1	1	1	1	1
10		1	1	1	1	1	1	1

$x_1 x_2 = 00$ column of flow table

Figure 15-19

		#1	#2	#3	#4	#5	#6	#7
$x_1 x_2$	$y_1 y_2$	00	00	00	00	00	00	00
00		00	00	00	00	00	00	00
01		00	00	00	00	11	00	10
11		00	01	10	01	10	01	10
10		00	00	00	11	00	01	00

$x_1 x_2 = 00$ column of Y-map

Figure 15-20

In option #1, the transition from the $y_1 y_2 = 11$ row to the $y_1 y_2 = 00$ row is accomplished by a noncritical race. In option #2, it is accomplished by a cycle: $y_1 y_2 = 11$ to 01 to 00. Option #4 is similar to option #2 except that the transition from the $y_1 y_2 = 10$ row to the $y_1 y_2 = 00$ row, instead of being direct, is accomplished by a cycle: $y_1 y_2 = 10$ to 11 to 01 to 00. Option #6 is similar to option #2 except that a noncritical race enters into the transition from the $y_1 y_2 = 10$ row.

Options #3, #5, and #7 are similar in type to options #2, #4, and #6 respectively.

The completed flow table with secondary assignment, using option #1, is shown in Fig. 15-21, with the associated Y -map and secondary excitation expressions.

		x_1x_2				
y_1y_2		00	01	11	10	
00		①	2	⑦	4	a
01		1	②	3	⑥	c
11		1	⑤	③	6	b
10		1	5	7	④	d

Flow table with
secondary assignment

		x_1x_2				
y_1y_2		00	01	11	10	
00		00	01	00	10	
01		00	01	11	01	
11		00	11	11	01	
10		00	11	00	10	

Y -map

Figure 15-21

When various options are possible, as in the example above, or when alternative assignments are possible, they should all be investigated, since at this stage of design there is no way of knowing which one will lead to the most economical solution. The economy of both the secondary excitation and output circuits can be affected by the choice of assignment and the choice of optional transitions.

PROBLEMS

1. Merge the primitive flow table in Fig. 15-22.

		x_1x_2				
		00	01	11	10	z
①		4	2	—	0	0
—		—	②	5	0	0
8		③	—	5	0	0
1		④	—	7	1	1
8		4	2	⑤	0	0
8		3	⑥	—	1	1
1		4	—	⑦	1	1
⑧		3	6	—	1	1

Figure 15-22

		x_1x_2				
		00	01	11	10	z
①		5	—	4	0	0
②		—	—	9	1	1
—		7	③	—	0	0
6		7	3	④	1	1
1		⑤	8	9	1	1
⑥		7	—	9	0	0
6		⑦	8	—	0	0
—		7	⑧	9	1	1
2		5	3	⑨	0	0

Figure 15-23

- *2. Merge the primitive flow table in Fig. 15-23.

16

Sequential Circuits IV

Utilization of Spare Secondary States

A secondary assignment for a three-row flow table can always be achieved with two secondary variables, although sometimes the “spare” fourth secondary state must be utilized to avoid critical races. This requirement occurs when there are transitions between all three pairs of rows, ab , ac , and bc , as illustrated in Fig. 16-1. No matter how secondary assignments are made, a transition will be required between two rows whose secondary state assignments differ in two variables.

For example, with the secondary assignment shown in the transition map in Fig. 16-2 rows a and c are each adjacent to row b , but are not adjacent to each other. However, transitions between rows a and c can be made without introducing critical race conditions: cycles through the spare $y_1y_2 = 01$

x_1x_2					
00	01	11	10		
①	2	3	④	a	
5	②	⑥	⑦	b	
⑤	⑧	③	4	c	

Figure 16-1

y_1		y_2	
0	1	0	1
a	b		
d	c		

Figure 16-2

secondary state, arbitrarily labeled d , can be prescribed. The flow table with this secondary assignment is shown in Fig. 16-3, with the associated Y -map.

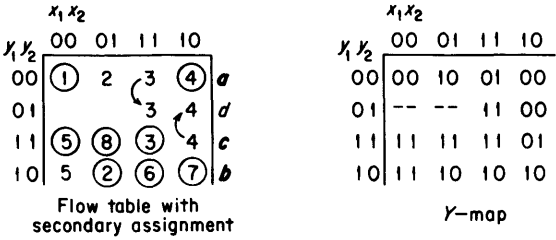


Figure 16-3

Two other secondary assignments for the same problem are shown in Figs. 16-4 and 16-5, one in which rows b and c are not adjacent, and the other in which rows a and b are not adjacent.

The three secondary assignments shown (Figs. 16-3, 16-4, 16-5) will, in general, lead to different solutions.

A secondary assignment, without critical races, for a four-row flow table can not always be achieved with two secondary variables, and three secondaries may be required. With three secondaries, eight secondary states are available: four for assignment to the four rows of the table, and four as spares.

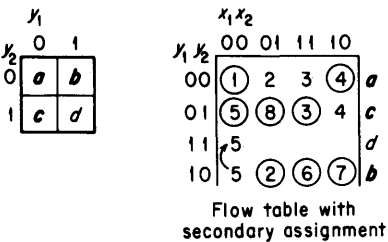


Figure 16-4

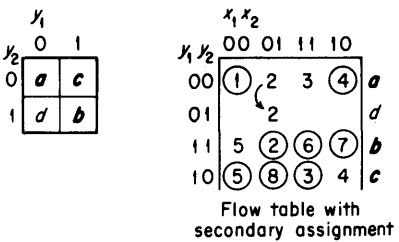


Figure 16-5

An example of a four-row flow table requiring three secondaries is shown in Fig. 16-6. This example illustrates a “worst case” condition in which there are transitions between all six pairs of rows, ab , ac , ad , bc , bd , and cd . Three secondaries may be required, however, for four-row flow tables in which there are transitions between as few as three of the six pairs of rows, as Fig. 16-7 illustrates.

x_1x_2		00	01	11	10	
①	2	3	4	a		
5	②	⑥	⑦	b		
⑤	8	6	④	c		
1	⑧	③	7	d		

Figure 16-6

$x_1 x_2$		00	01	11	10	
①	2	3	4	a		
1	②	⑤	⑥	b		
1	⑦	③	⑧	c		
1	⑨	⑩	④	d		

Figure 16-7

Sometimes it may appear, at first, that additional secondaries are needed when actually this is not the case. For example, in Fig. 16-8, analysis of the left three columns of the flow table shows transitions between rows a and b , b and c , c and d , and a and d , and the transition map so far represents a satisfactory secondary assignment.

The $x_1x_2 = 10$ column contains a transition from d to c , for which the secondary assignment shown is still satisfactory. However, this column also contains a transition from a to c , which, with the assignment shown, involves a change of two secondary variables. A critical race can be avoided without the use of an additional secondary, however, by the cycle adc being prescribed for this transition.

The following analysis of secondary assignment patterns is intended to give the reader an appreciation for the number of ways in which secondary assignments can be made, and to give him a feel for the types of variations possible. The patterns are numbered for reference purposes only, and are presented for analysis. It should not be inferred that in a particular problem, some pattern should be “chosen.” The secondary assignment should most generally be “tailor made” for each problem; the reader, having studied the patterns, should have a more thorough understanding of the variations possible.

When three secondaries are required for a four-row flow table, there are seventy ways of selecting four out of eight secondary states for assignment to four rows

$$\left({}_8C_4 = \frac{8!}{4!4!} = 70 \right)$$

PatternWord descriptionExample

#1

w and x differ in one variable
 x and y differ in one variable
 y and z differ in one variable
 z and w differ in one variable

		$y_1 y_2$			
y_3		00	01	11	10
	0	w	x		
	1	z	y		

Figure 16-9

#2

x and w differ in one variable
 x and y differ in one variable
 x and z differ in one variable

		$y_1 y_2$			
y_3		00	01	11	10
	0	w	x	y	
	1		z		

Figure 16-10

#3

w and x differ in one variable
 x and y differ in one variable
 y and z differ in one variable
 z and w differ in three variables

		$y_1 y_2$			
y_3		00	01	11	10
	0	w	x	y	
	1			z	

Figure 16-11

#4

x and w differ in one variable
 x and y differ in one variable
 x and z differ in three variables

		$y_1 y_2$			
y_3		00	01	11	10
	0	w	x	y	
	1				z

Figure 16-12

#5

w and x differ in one variable
 y and z differ in one variable
 w and y differ in three variables
 x and z differ in three variables

		$y_1 y_2$			
y_3		00	01	11	10
	0	w	x		
	1			y	z

Figure 16-13

#6

All pairs of states differ in two variables

		$y_1 y_2$			
y_3		00	01	11	10
	0	w		x	
	1		y		z

Figure 16-14

Analysis shows that each of the seventy combinations falls into one of six different patterns. A word description of each pattern, with an example of each in a transition map, is given on p. 220. The selected states are arbitrarily labeled *w*, *x*, *y*, and *z* to aid in the word descriptions.

The application of pattern #1 is limited: it can be used for any¹ four-row flow table in which there are transitions between five or fewer of the six pairs of rows, but it can be used only for *some* worst case conditions. Patterns #2 through #6 can be used for any¹ four-row flow table.

For each combination, there are twenty-four permutations of row assignment

$$({}_4P_4 = 4! = 24)$$

that is, there are twenty-four ways of assigning four selected secondary states to four rows. Analysis shows that for each pattern, certain row assignments lead to solutions that are equivalent with secondaries interchanged. The number of nonequivalent row assignments for each pattern follows:

<i>Pattern</i>	<i>Number of nonequivalent row assignments</i>
#1	3
#2	4
#3	12
#4	12
#5	6
#6	1
	<hr/> 38

If three secondaries are required for a four-row flow table, there are thus thirty-eight secondary assignments that will, in general, lead to different solutions (thirty-five in cases in which pattern #1 is not applicable).

Examples of the thirty-eight assignments are shown in Fig. 16-15. The pattern variations in the fourth example of pattern #2, and in the last six examples of patterns #3 and #4, are made to retain the $y_1y_2y_3 = 000$ assignment for row *a*.

Each pattern will now be examined in more detail and illustrated by a flow table example. For each pattern, a resulting flow table with secondary assignment, and its associated *Y*-map, will be shown for study.

¹Exceptions can occur when cycles are required as part of the original circuit specifications; see section on *Transient Outputs; Cyclic Specifications*. Cyclic specifications are not considered in the discussion that follows.

Pattern

#1	#2	#3	#4	#5	#6																																																
$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td></td><td></td></tr><tr><td>d</td><td>c</td><td></td><td></td></tr></table> 1	a	b			d	c			$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>c</td><td></td></tr><tr><td>d</td><td></td><td></td><td></td></tr></table> 1	a	b	c		d				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>c</td><td></td></tr><tr><td></td><td></td><td>d</td><td></td></tr></table> 1	a	b	c				d		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>c</td><td></td></tr><tr><td></td><td></td><td>d</td><td></td></tr></table> 1	a	b	c				d		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td></td><td></td></tr><tr><td></td><td></td><td>c</td><td>d</td></tr></table> 1	a	b					c	d	$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td></td><td>b</td><td></td></tr><tr><td></td><td>c</td><td></td><td>d</td></tr></table> 1	a		b			c		d
a	b																																																				
d	c																																																				
a	b	c																																																			
d																																																					
a	b	c																																																			
		d																																																			
a	b	c																																																			
		d																																																			
a	b																																																				
		c	d																																																		
a		b																																																			
	c		d																																																		
$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td></td><td></td></tr><tr><td>c</td><td>d</td><td></td><td></td></tr></table> 1	a	b			c	d			$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>b</td><td></td></tr><tr><td></td><td>d</td><td></td><td></td></tr></table> 1	a	c	b			d			$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>d</td><td></td></tr><tr><td></td><td></td><td></td><td>c</td></tr></table> 1	a	b	d					c	$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>d</td><td></td></tr><tr><td></td><td></td><td></td><td>c</td></tr></table> 1	a	b	d					c	$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td></td><td></td><td></td></tr><tr><td></td><td>d</td><td></td><td>c</td></tr></table> 1	a					d		c									
a	b																																																				
c	d																																																				
a	c	b																																																			
	d																																																				
a	b	d																																																			
			c																																																		
a	b	d																																																			
			c																																																		
a																																																					
	d		c																																																		
$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td></td><td></td></tr><tr><td>d</td><td>b</td><td></td><td></td></tr></table> 1	a	c			d	b			$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>b</td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table> 1	a	d	b						$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>b</td><td></td></tr><tr><td></td><td></td><td>d</td><td></td></tr></table> 1	a	c	b				d		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>b</td><td></td></tr><tr><td></td><td></td><td>d</td><td></td></tr></table> 1	a	c	b				d		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td></td><td></td></tr><tr><td></td><td></td><td>b</td><td>d</td></tr></table> 1	a	c					b	d									
a	c																																																				
d	b																																																				
a	d	b																																																			
a	c	b																																																			
		d																																																			
a	c	b																																																			
		d																																																			
a	c																																																				
		b	d																																																		
	$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>b</td><td></td></tr><tr><td>d</td><td></td><td></td><td></td></tr></table> 1	a	c	b		d				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>d</td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	c	d				b		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>d</td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	c	d				b		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>d</td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	c	d				b		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td></td><td></td></tr><tr><td></td><td></td><td>d</td><td>b</td></tr></table> 1	a	c					d	b								
a	c	b																																																			
d																																																					
a	c	d																																																			
		b																																																			
a	c	d																																																			
		b																																																			
a	c	d																																																			
		b																																																			
a	c																																																				
		d	b																																																		
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>b</td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table> 1	a	d	b						$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>b</td><td></td></tr><tr><td></td><td></td><td></td><td>c</td></tr></table> 1	a	d	b					c	$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td></td></tr><tr><td></td><td></td><td>b</td><td>c</td></tr></table> 1	a	d					b	c	$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td></td></tr><tr><td></td><td></td><td>b</td><td>c</td></tr></table> 1	a	d					b	c																
a	d	b																																																			
a	d	b																																																			
			c																																																		
a	d																																																				
		b	c																																																		
a	d																																																				
		b	c																																																		
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>c</td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	d	c				b		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>c</td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	d	c				b		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	d					b		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td></td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	d					b																	
a	d	c																																																			
		b																																																			
a	d	c																																																			
		b																																																			
a	d																																																				
		b																																																			
a	d																																																				
		b																																																			
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>d</td><td></td></tr><tr><td>b</td><td></td><td></td><td></td></tr></table> 1	a	c	d		b				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td></td><td>b</td></tr><tr><td></td><td></td><td>d</td><td></td></tr></table> 1	a	c		b			d																																			
a	c	d																																																			
b																																																					
a	c		b																																																		
		d																																																			
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>c</td><td></td></tr><tr><td>b</td><td></td><td></td><td></td></tr></table> 1	a	d	c		b				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td>b</td></tr><tr><td></td><td></td><td>c</td><td></td></tr></table> 1	a	d		b			c																																			
a	d	c																																																			
b																																																					
a	d		b																																																		
		c																																																			
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>d</td><td></td></tr><tr><td>c</td><td></td><td></td><td></td></tr></table> 1	a	b	d		c				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td>c</td></tr><tr><td></td><td></td><td>b</td><td></td></tr></table> 1	a	d		c			b																																			
a	b	d																																																			
c																																																					
a	d		c																																																		
		b																																																			
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>b</td><td></td></tr><tr><td>c</td><td></td><td></td><td></td></tr></table> 1	a	d	b		c				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td></td><td>b</td></tr><tr><td></td><td>d</td><td>c</td><td>b</td></tr></table> 1	a	d		b		d	c	b																																		
a	d	b																																																			
c																																																					
a	d		b																																																		
	d	c	b																																																		
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>d</td><td>b</td><td></td></tr><tr><td>c</td><td></td><td></td><td></td></tr></table> 1	a	d	b		c				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td></td><td></td><td></td></tr><tr><td></td><td>d</td><td>c</td><td>b</td></tr></table> 1	a					d	c	b																																		
a	d	b																																																			
c																																																					
a																																																					
	d	c	b																																																		
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>b</td><td>c</td><td></td></tr><tr><td>d</td><td></td><td></td><td></td></tr></table> 1	a	b	c		d				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td></td><td></td><td></td></tr><tr><td></td><td>c</td><td>d</td><td>b</td></tr></table> 1	a					c	d	b																																		
a	b	c																																																			
d																																																					
a																																																					
	c	d	b																																																		
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>b</td><td></td></tr><tr><td>d</td><td></td><td></td><td></td></tr></table> 1	a	c	b		d				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td></td><td></td><td></td></tr><tr><td></td><td>c</td><td>d</td><td>b</td></tr></table> 1	a					c	d	b																																		
a	c	b																																																			
d																																																					
a																																																					
	c	d	b																																																		
		$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td>c</td><td>b</td><td></td></tr><tr><td>d</td><td></td><td></td><td></td></tr></table> 1	a	c	b		d				$x_1 x_2$ y_3 00 01 11 10 0 <table><tr><td>a</td><td></td><td></td><td></td></tr><tr><td></td><td>d</td><td>b</td><td>c</td></tr></table> 1	a					d	b	c																																		
a	c	b																																																			
d																																																					
a																																																					
	d	b	c																																																		

Examples of three-variable secondary assignments for four-row flow tables

Figure 16-15

Pattern #1

In the transition map of pattern #1 (Fig. 16-16), the four spare secondary states are labeled arbitrarily, *e*, *f*, *g*, and *h*.

Transitions between states *w* and *x*, *x* and *y*, *y* and *z*, and *z* and *w* are direct, since they involve the change of only one variable. Critical races in the transitions between states *w* and *y* can be avoided by the utilization of the spare states. The cycle

$$wefhy$$

or the cycle

$$weghy$$

or the cycle with noncritical race

$$wehy$$

can be prescribed. If a race from *e* to *h* is prescribed, *f* and *g* must also be directed to *h*; if a race from *h* to *e* is prescribed, *f* and *g* must be directed to *e*.

The three variations can be summarized by the notation

$$we(fg)hy$$

which signifies that the transitions between *e* and *h* may involve a non-critical race or a cycle through *f* or *g*. The notations apply in both directions; for example, for transitions from *w* to *y* and from *y* to *w*.

Transitions between states *x* and *z* can be similarly prescribed as follows:

$$xf(eh)gz$$

The limitation on the application of pattern #1 is that transitions between states *w* and *y* and between *x* and *z* cannot occur in the same column because of the conflicts that would result in the direction, in that column, of the spare states utilized in both transitions. Therefore, if there are transitions between two pairs of rows in the same column, the assignment of states *w* and *y* to one pair, and states *x* and *z* to the other pair, cannot be allowed. If such an assignment cannot be avoided, pattern #1 cannot be used. The following example illustrates this condition.

In the $x_1x_2 = 00$ column of Fig. 16-17, there are transitions between rows *a* and *d*, and between *b* and *c*. Therefore, the assignment of states *w* and *y* to rows *a* and *d*, and the assignment of states *x* and *z* to rows *b* and *c*, or vice versa, is not allowed. In the $x_1x_2 = 01$ column, there are transitions between rows *a* and *b*, and between *c* and *d*. Therefore, the assignment of states *w* and *y* to rows *a* and *b*, and the assignment of states *x* and *z* to rows *c* and *d*, or vice versa, is not allowed. In the $x_1x_2 = 11$

		x_1x_2			
		00	01	11	10
y_3	0	<i>w</i>	<i>x</i>	<i>f</i>	<i>e</i>
	1	<i>z</i>	<i>y</i>	<i>h</i>	<i>g</i>

Figure 16-16

		x_1x_2			
		00	01	11	10
	1	2	3	4	<i>a</i>
	5	6	7		<i>b</i>
	8	6	4		<i>c</i>
	9	8	3	7	<i>d</i>

Figure 16-17

column, there are transitions between the same pairs of rows as in the $x_1x_2 = 00$ column. In the $x_1x_2 = 10$ column, there are transitions between rows a and c and between b and d . Therefore the assignment of states w and y to rows a and c , and the assignment of states x and z to rows b and d , or vice versa, is not allowed.

There is thus no assignment that will avoid transitions between states w and y and between x and z in the same column, and pattern #1 cannot be used for this flow table. In the example that follows, pattern #1 can be used.

The $x_1x_2 = 00$ and $x_1x_2 = 01$ columns in Fig. 16-18 are the same as in the preceding example, and therefore the assignment of states w and y to rows a and d , and the assignment of states x and z to rows b and c , or vice versa is not allowed; and the assignment of states w and y to rows a and b , and the assignment of states x and z to rows c and d , or vice versa, is also not allowed.

$x_1 x_2$						
		00	01	11	10	
x_3	0	① 2	③ 4			a
	5	② 6	⑦ 8			b
	⑤ 8	⑨ 4				c
	1	⑧ 6	⑩ 10			d

Figure 16-18

x_1x_2					
		00	01	11	10
x_3	0	a	b	f	e
	1	d	c	h	g

Figure 16-19

All six possible transitions occur in the flow table, but the remaining two transitions occur in different columns: the transition between rows b and d occurs in the $x_1x_2 = 11$ column, and the transition between rows a and c occurs in the $x_1x_2 = 10$ column. Therefore, states w and y can be assigned to rows b and d , and states x and z can be assigned to rows a and c , or vice versa, and pattern #1 is thus applicable. The assignment in Fig. 16-19 is chosen. Transitions between rows a and c are prescribed by

$$ae(fg)hc$$

and transitions between rows b and d are prescribed by

$$bf(eh)gd$$

A resulting flow table with secondary assignment, and its associated Y-map, are shown in Fig. 16-20. The cycle $dgefb$ is chosen for the d to b transition in the $x_1x_2 = 11$ column. The cycle with noncritical race $aehc$ is chosen for the a to c transition in the $x_1x_2 = 10$ column. The choices of optional transitions in this and the following examples are arbitrary, and are selected to illustrate the various types of transitions possible.

The flow table in Fig. 16-21 will be used as a running example in the

examination of patterns #2 through #6. The assignment illustrating each pattern is chosen arbitrarily.

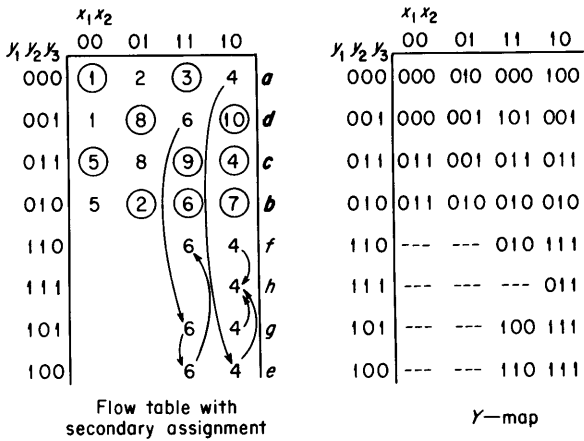


Figure 16-20

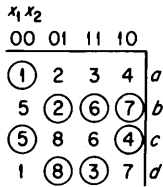


Figure 16-21

Pattern #2

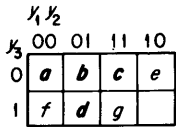


Figure 16-22

Transitions between rows *b* and *a*, *b* and *c*, and *b* and *d* are direct. The remaining transitions are prescribed by

- aec*
- afd*
- cgd*

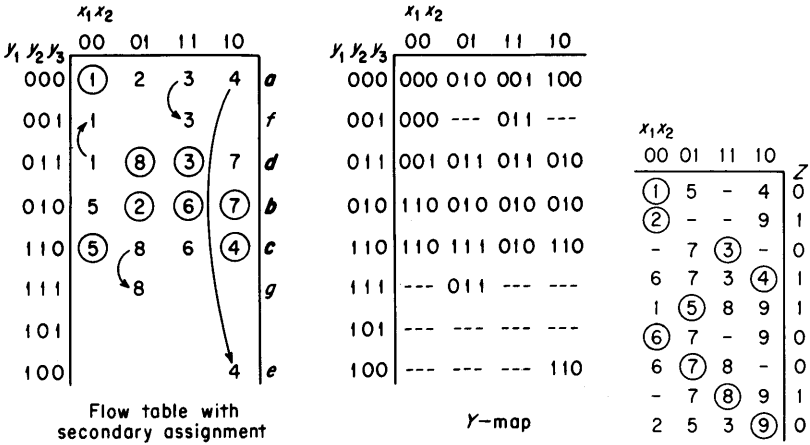


Figure 16-23

Pattern #3

$x_1 x_2$					
		00	01	11	10
y_3	0	<i>a</i>	<i>b</i>	<i>c</i>	<i>e</i>
	1	<i>f</i>	<i>g</i>	<i>d</i>	<i>h</i>

Figure 16-24

Transitions between rows *a* and *b*, *b* and *c*, and *c* and *d* are direct. The remaining transitions are prescribed by

$$\begin{aligned}
 &aec \\
 &bgd \\
 &a(ef)hd \text{ or } af(gh)d
 \end{aligned}$$

If the *a* to *d* transition *afhd* is chosen, there can, of course, be no conflicts. However, the spare states *e* and *g* can be utilized in more than one type of transition without conflicts. For example, consider the transitions involving the spare state *e*: *aec* and *ae hd*. Transitions from *a* to *c* and from *a* to *d* cannot occur in the same column. Transitions from *c* to *a* and from *d* to *a*, even if they do occur in the same column, cause no conflict, *e* being directed to *a* in both transitions. Also, of course, transitions from *a* and to *a* cannot occur in the same column.

When there are two types of transitions, with a row of a flow table common to both, for example, row *a* in the preceding example, the same spare states can be utilized in both transitions without conflict.

If two types of transitions have no rows of a flow table in common, the

same spare states can be utilized in both transitions, without conflict, only if the transitions do not occur in the same column (see examples in the discussion of pattern #1).

When there is a choice of transitions between two rows, the one selected for any particular column is independent of that chosen in any other column. With pattern #3, for example, *ae**hd* might be chosen for the *a* to *d* transition in one column, and *af**gd* might be chosen in another column.

In the flow table in Fig. 16-25, the cycle with noncritical race *d**f**a* is chosen for the *d* to *a* transition in the $x_1x_2 = 00$ column; and the cycle *ae**hd* is chosen for the *a* to *d* transition in the $x_1x_2 = 11$ column.

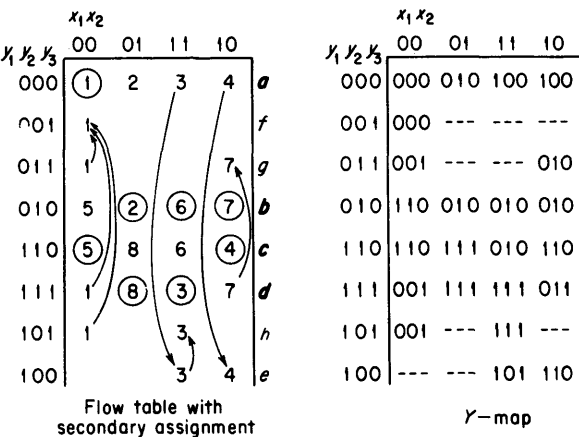


Figure 16-25

Pattern #4

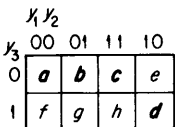


Figure 16-26

Transitions between rows *a* and *b*, and *b* and *c* are direct. The remaining transitions are prescribed by

aec

a(*ef*)*d*

c(*eh*)*d*

bg(*fh*)*d*

Note that the spare states e, f and h can be utilized in more than one type of transition, similar to the spare states in pattern #3.

In the flow table in Fig. 16-27, the noncritical race da is chosen for the d to a transition in the $x_1x_2 = 00$ column; the cycle aed is chosen for the a to d transition in the $x_1x_2 = 11$ column; the cycle ced is chosen for the c to d transition in the $x_1x_2 = 01$ column; and the cycle with noncritical race dgb is chosen for the d to b transition in the $x_1x_2 = 10$ column.

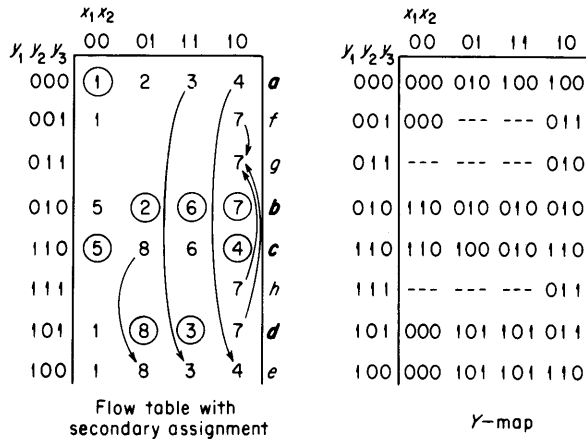


Figure 16-27

Pattern #5

	x_1x_2			
	00	01	11	10
y_3 0	<i>a</i>	<i>b</i>	<i>h</i>	<i>e</i>
1	<i>f</i>	<i>g</i>	<i>c</i>	<i>d</i>

Figure 16-28

Transitions between rows a and b , and c and d are direct. The remaining transitions are prescribed by

$$\begin{array}{l}
 a(ef)d \\
 b(gh)c \\
 aehc \quad \text{or} \quad afgc \\
 bhed \quad \text{or} \quad bgfd
 \end{array}$$

All spare states can be utilized in more than one type of transition, subject to the following restriction: since transitions between rows a and c and between rows b and d involve no rows in common, if both types of

transitions occur in the same column, the same spare states cannot be utilized in both, and the choice of transitions must be restricted to

$a(ef)d$

$b(gh)c$

$aehc$

$bgfd$

or

$a(ef)d$

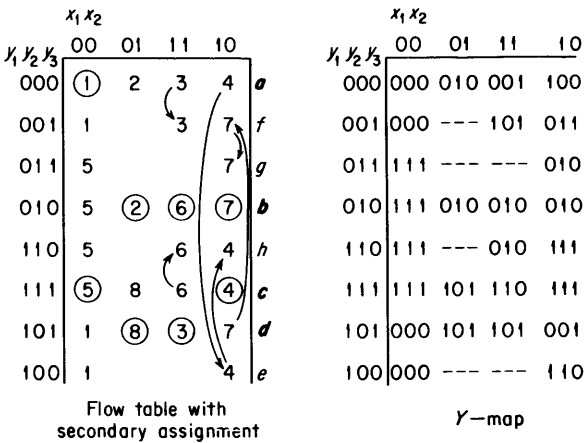
$b(gh)c$

$afgc$

$bhed$

In the running example, the above restriction must be observed since, in the $x_1x_2 = 10$ column, there is a transition from row a to c , and from d to b .

In the flow table in Fig. 16-29, the noncritical races da and bc are chosen for the respective d to a and b to c transitions in the $x_1x_2 = 00$ column; the cycles afd and chb are chosen for the respective a to d and c to b transitions in the $x_1x_2 = 11$ column; and the cycles $aehc$ and $dfgb$ are chosen for the respective a to c and d to b transitions in the $x_1x_2 = 10$ column.



$a(eg)b$ $a(fg)c$ $a(ef)d$ $b(gh)c$ $b(eh)d$ $c(fh)d$

Note that all spare states can be utilized in more than one type of transition. Also note that noncritical races can be used for all transitions.

In the flow table in Fig. 16-31, the cycles dea and bgc are chosen for the respective d to a and b to c transitions in the $x_1x_2 = 00$ column. The cycle aeb and the noncritical race cd are chosen for the respective a to b and c to d transitions in the $x_1x_2 = 01$ column. The noncritical races ad and cb are chosen for the respective a to d and c to b transitions in the $x_1x_2 = 11$ column. The noncritical race ac and the cycle deb are chosen for the respective a to c and d to b transitions in the $x_1x_2 = 10$ column.

		x_1x_2				
$x_1x_2x_3$		00	01	11	10	
000	①	2	3	4	a	000 000 100 101 011
001		8	3	4	f	001 --- 101 101 011
011	⑤	8	6	④	c	011 011 101 110 011
010	5		6	4	g	010 011 --- 110 011
110	5	②	⑥	⑦	b	110 010 110 110 110
111		8	6		h	111 --- 101 110 ---
101	1	⑧	③	7	d	101 100 101 101 100
100	1	2	3	7	e	100 000 110 101 110

Flow table with
secondary assignment

Y -map

Figure 16-31

Assignment of Multiple Secondary States to a Row

The discussion of secondary assignments so far has been limited to assignments in which each row of a flow table is assigned *one* secondary state. When the use of spare secondary states is required, however, rows of a flow table can be assigned more than one secondary state, and this type of secondary assignment will now be examined.

A secondary assignment for the three-row flow table previously examined (Fig. 16-32), requires the use of the spare fourth secondary state. In a previous example, the secondary assignment in Fig. 16-33 was made. In the transitions between rows a and c , critical races were avoided by cycles through the spare $y_1y_2 = 01$ secondary state, labeled d , being prescribed.

		x_1x_2			
		00	01	11	10
y_1	0	(1)	2	3	(4)
	1	5	(2)	(6)	(7)
		(5)	(8)	(3)	4

Figure 16-32

		y_1	
		0	1
y_2	0	a	b
	1	d	c

Figure 16-33

The preceding assignment can be modified by the assignment of the spare $y_1y_2 = 01$ state to row a or row c , which still avoids critical races in the transitions between rows a and c .

EXAMPLE:

When more than one secondary state is assigned to the same row, rows with equivalent stable states are created in the flow table with secondary assignment. Subscripts are used to differentiate these rows and equivalent stable states for transition identification. The output conditions of equivalent stable states are the same, and the selection of equivalent states for a particular transition is based fundamentally on the avoidance of critical races. Refer to Fig. 16-34.

Transitions from b to a are directed to row a_1 ; transitions from c to a are directed to a_2 ; transitions from a_2 to b cycle through a_1 ; and transitions from a_1 to c cycle through a_2 .

The flow table with secondary assignment, and its associated Y -map, are shown in Fig. 16-35.

		x_1x_2			
		00	01	11	10
y_1	0	(1) ₁	2	3	(4) ₁
	1	(1) ₂	2	3	(4) ₂
		(5)	(8)	(3)	4 ₂
		5	(2)	(6)	(7)

Flow table with secondary assignment

		x_1x_2			
		00	01	11	10
y_1	0	00	10	01	00
	1	01	00	11	01
		11	11	11	01
		10	11	10	10

Y -map

Figure 16-34

Figure 16-35

Subscripts are used on unstable state numbers where necessary to specify to which of equivalent stable states a transition occurs.

Figures 16-36 and 16-37 show another assignment for the same problem, with the $y_1y_2 = 01$ secondary state assigned to row c instead of row a .

		$x_1 x_2$	
		00	01
y_1	0	a	b
y_2	1	c_1	c_2

Figure 16-36

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00	(1) 2	3 ₁ (4)	a	
	01	(5 ₁) (8 ₁) (3 ₁)	c_1		
	11	(5 ₂) (8 ₂) (3 ₂)	c_2		
	10	5 ₂ (2) (6) (7)	b		

Flow table with secondary assignment

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00	00	10	01	00
	01	01	01	01	00
	11	11	11	11	01
	10	11	10	10	10

Y-map

Figure 16-37

The assignment of multiple secondary states to a row can be modified by the replacement of some equivalent stable states with the corresponding unstable states. For example, the flow table in Fig. 16-37 could be modified as shown in Fig. 16-38.

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00	(1) 2	3 ₁ (4)	a	
	01	(5) 8	(3 ₁)	c_1	
	11	5 (8) (3 ₂)	c_2		
	10	5 (2) (6) (7)	b		

Flow table with secondary assignment

Figure 16-38

When three secondaries are required for a four-row flow table, and the assignment of more than one secondary state to a row is considered, many additional patterns become possible. Some examples follow (Figs. 16-39 to 16-43), in which all eight secondary states are assigned to rows.

Notice that the pattern in Fig. 16-43 differs from the others in that no rows with equivalent stable states are adjacent, but each row is adjacent to one of each type of nonequivalent row, so that all transitions are direct.

		$x_1 x_2$			
		00	01	11	10
y_3	0	a_1	b_1	c_1	c_2
	1	a_2	b_2	d_1	d_2

Figure 16-39

		$x_1 x_2$			
		00	01	11	10
y_3	0	a_1	b	c_1	c_2
	1	a_2	d_1	d_2	d_3

Figure 16-40

		$x_1 x_2$			
		00	01	11	10
y_3	0	a	b	c_1	c_2
	1	d_1	d_2	d_3	d_4

Figure 16-41

		$x_1 x_2$			
		00	01	11	10
y_3	0	a_1	b	c	a_3
	1	a_2	d_1	d_2	d_3

Figure 16-42

		$x_1 x_2$			
		00	01	11	10
y_3	0	a_1	b_1	c_1	d_1
	1	c_2	d_2	a_2	b_2

Figure 16-43

Utilization of Spare Secondary States— Summary

The utilization of spare secondary states for three- and four-row flow tables has been discussed in some detail. The many examples studied are to make the reader aware of the variations possible: the selection of patterns; the selection of assignments for each pattern; and the selection of optional transitions and modifications for each assignment. The concepts discussed can be extended to flow tables of more than four rows.

The ultimate selection of a flow table with secondary assignment is most generally based on circuit economy. When circuit economy is the criterion, the comparison of different flow tables with secondary assignment cannot be made directly; the associated Y -maps and Z (output)-maps must be obtained, the secondary excitation and output expressions obtained from the maps, and the resulting total circuits compared.

Blank entries in a flow table result in corresponding optional entries in the associated Y -map. Optional entries are, of course, generally desirable from a circuit economy standpoint.

Having made a "trial" assignment and obtained the corresponding Y -map and Z -map, it might be observed that better groups, i.e. simpler expressions, could be obtained if particular map entries were 1's instead of 0's, or vice versa. Making such a change might necessitate the assignment of multiple secondary states to a row.

One should also take the fullest advantage of the generally wide selection of optional transitions and modifications for an assignment. For example, in Fig. 16-41, the possible transitions from d_1 to c are:

- (a) $d_1 d_4 c_2$
- (b) $d_1 d_4 d_3 c_1$
- (c) $d_1 d_2 d_3 c_1$
- (d) $d_1 d_2 d_3 d_4 c_2$
- (e) $d_1 d_4 c_1$, with d_3 directed to c_1
- (f) $d_1 d_2 d_3 c_2$, with d_4 directed to c_2
- (g) $d_1 d_3 c_1$, with d_2 directed to d_3 ; and d_4 directed to d_3 , c_1 or c_2
- (h) $d_1 d_3 c_2$, with d_2 directed to d_3 ; and d_4 directed to c_2 .

Note that in transitions (e) through (h), circuit action may terminate in either c_1 or c_2 .

The choice of secondary assignment may also be based on the speed of transition. For example, pattern #6, with noncritical races prescribed for all transitions, might be chosen if it were desired that all transition times be short; whereas pattern #1 might be chosen if a long time delay were desired for a particular transition.

PROBLEMS

1. Make a secondary assignment for the flow table in Fig. 16-44, draw the associated Y -map, and obtain the secondary excitation expressions.

$x_1 x_2$				
00	01	11	10	
①	2	5	⑧	
③	4	⑤	7	
3	②	⑥	7	
1	④	5	⑦	

Figure 16-44

- *2. Make a secondary assignment for the flow table in Fig. 16-45, draw the associated Y -map, and obtain the secondary excitation expressions.

$x_1 x_2$				
00	01	11	10	
①	②	4	7	
6	③	④	8	
⑥	3	⑤	⑦	
6	2	4	⑧	

Figure 16-45

3. Obtain the Y -map from the flow table in Fig. 16-46.

		$x_1 x_2$			
		00	01	11	10
$x_1 x_2 x_3$	000	① ₁	3	6	8
	001	① ₂	3	6 ₂	8
	011				7
	010	2	③	⑤	⑦
	110	②	4 ₁	5	⑧
	111	1	④ ₁	⑥ ₁	7
	101	1 ₂	④ ₂	⑥ ₂	7
	100				8

Figure 16-46

- *4. Obtain the Y -map from the flow table in Fig. 16-47.

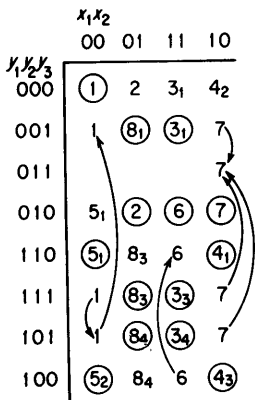


Figure 16-47

5. Obtain the secondary excitation expressions from the Y -map in Fig. 16-48. Find all optimum solutions.

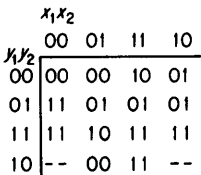


Figure 16-48

*6. Obtain the secondary excitation expressions from the Y -map in Fig. 16-49. Find all optimum solutions.

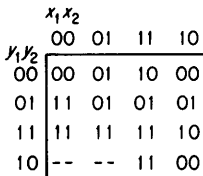


Figure 16-49

17

Sequential Circuits V

Z-Map

The output expressions are read from the Z-map. The Z-map is obtained from the flow table with secondary assignment and the primitive flow table: The output state for each stable state is identified in the primitive flow table, and the location, in the Z-map, of the output state is identified in the flow table with secondary assignment. Also, the actual state to state transitions are identified in the primitive flow table, this information being used in the assignment of output states for the unstable states.

Figure 17-1 shows a primitive flow table and a corresponding merged flow table with secondary assignment.

In constructing the Z-map, the output state for each stable state is entered in the map first. The output state for each stable state is found in the primitive flow table; the location, in the Z-map, of the output state corresponds to the location of the associated stable state in the flow table

$x_1 x_2$						
00	01	11	10		Z	
①	2	3	4	0		
-	②	5	4	0		
1	2	③	4	1		
1	2	5	④	1		
-	2	⑤	4	0		

Primitive flow table

$x_1 x_2$						
y	00	01	11	10		
0	①	2	③	4		
1	1	②	⑤	④		

Merged flow table
with secondary assignment

Figure 17-1

$x_1 x_2$						
y	00	01	11	10		
0	0		1			
1		0	0	1		

Partially-completed
 Z -map

Figure 17-2

with secondary assignment. The partially-completed Z -map is shown in Fig. 17-2.

In the assignment of output states for the unstable states, the following rules are observed:

- (1) If, in a transition, the states of an output for the initial and final stable states are the same, this same output state must be assigned for all unstable states involved in the transition. Transient changes in output state are thus prevented.
- (2) The output states for all unstable states not covered by rule 1 may be optional except that in all transitions involving a change in output state, the output must change state only once. The exception must be noted only when there are two or more unstable states involved in a transition; oscillatory changes of output states are thus prevented.

The preceding Z -map will now be completed. The flow table with secondary assignment indicates that unstable state 2 may be involved in a transition from stable state ① to ② or from ③ to ②. The ① to ② transition specifies no change in output state (the initial output state is 0, and the final output state is 0). If this transition does in fact occur, the output state must be 0 for unstable state 2. Reference to the primitive flow table shows that the ① to ② transition does occur; therefore the output state 0 must be assigned to unstable state 2.

The flow table with secondary assignment indicates that unstable state 4 may be involved in a transition from ① to ④ or from ③ to ④. The ③ to ④ transition is the critical one in this case; both the initial and final output states are 1. Reference to the primitive flow table shows that the ③ to ④ transition occurs; therefore, the output state must be 1 for unstable state 4.

The flow table with secondary assignment indicates that unstable state 1 may be involved in a transition from ② to ①, from ⑤ to ①, or from ④ to ①. The ② to ① and ⑤ to ① transitions both specify no change in

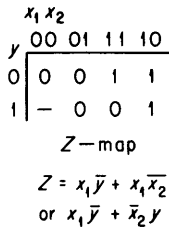


Figure 17-3

output state; if *either* transition occurs, the output state must be 0 for unstable state 1. The primitive flow table shows, however, that neither of these transitions occurs. Since unstable state 1 is not involved in a transition specifying no change in output state, the output state may be optional for unstable state 1. Figure 17-3 shows the completed Z-map and the output expressions.

Figure 17-4 shows an example with two outputs. The primitive flow table is not shown, but from it has been obtained the output state for each stable state; these output states appear in the partially-completed Z-map. All left-hand map entries define Z_1 ; all right-hand map entries define Z_2 . The primitive flow table also shows that all transitions indicated in the flow table with secondary assignment do occur.

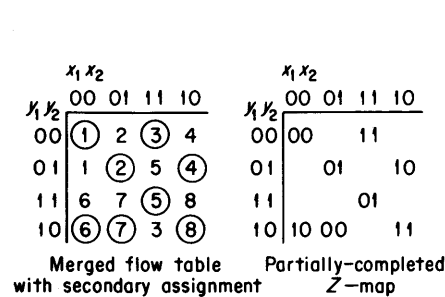


Figure 17-4

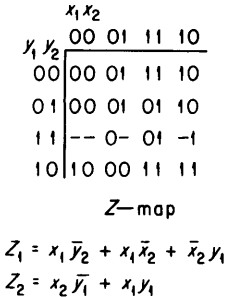


Figure 17-5

The Z-map will now be completed. Unstable state 2 is involved in transitions ① to ②, output states 00 → 01; and ③ to ②, output states 11 → 01. Examination of the two outputs independently shows that the ① to ②

Transition	Output states	
	Z_1	Z_2
① to ②	0 → 0	0 → 1
③ to ②	1 → 0	1 → 1

transition specifies no change in the Z_1 output state, requiring $Z_1 = 0$ for unstable state 2; and that the ③ to ② transition specifies no change in the Z_2 output state, requiring $Z_2 = 1$ for unstable state 2. Therefore, $Z_1 Z_2 = 01$ is required for unstable state 2.

The output state requirement, $Z_1 Z_2 = 10$, for unstable state 4 is similarly determined: the ① to ④ transition requires $Z_2 = 0$, and the ③ to ④ transition requires $Z_1 = 1$. Unstable state 1 requires the output state

$Z_1Z_2 = 00$, the ② to ① transition requiring $Z_1 = 0$, and the ④ to ① transition requiring $Z_2 = 0$. Unstable state 5 requires the output state $Z_1Z_2 = 01$, the ② to ⑤ transition establishing both the Z_1 and Z_2 output states. The output states for unstable state 6 can be optional, since in the ⑤ to ⑧ transition, both outputs change state. The ⑤ to ⑦ transition requires the output state $Z_1 = 0$ for unstable state 7, whereas Z_2 can be optional since it changes state. The ⑤ to ⑧ transition requires the output state $Z_2 = 1$ for unstable state 8, whereas Z_1 can be optional since it changes state. Unstable state 3 requires the output state $Z_1Z_2 = 11$.

The completed Z-map and the output expressions are shown in Fig. 17-5.

Sometimes there may be a requirement that, if a transition involves a multiple output change, all output state changes are to occur simultaneously. For example, with this requirement, in the ⑤ to ⑧ transition in the preceding example, the output states for unstable state 6 would be restricted to either $Z_1Z_2 = 01$ or $Z_1Z_2 = 10$.

Timing considerations may sometimes take precedence over circuit economy, and definite output states may be assigned in place of optional ones. For example:

If it is desired that all outputs be of as short a duration as possible, all optional entries can be replaced by 0's;

If it is desired that all outputs be of as long a duration as possible, all optional entries can be replaced by 1's;

If it is desired that all output changes occur as soon as possible, all optional entries can be replaced by the output entries for the corresponding final stable states;

If it is desired that all output changes occur as late as possible, all optional entries can be replaced by the output entries for the corresponding initial stable states.

Such timing considerations may, of course, apply to particular transitions only.

Some examples of output state assignments for unstable states involved in cycles and races follow.

Cycle—No Change in Output State

		x_1x_2			
x_1x_2		00	01	11	10
x_1x_2	00	①			
	01	1			
	11	1			
	10	1	②		

Flow table with secondary assignment

		x_1x_2			
x_1x_2		00	01	11	10
x_1x_2	00	0			
	01	0			
	11	0			
	10	0	0		

Z-map

Figure 17-6

$Z = 0$ must be assigned to all unstable states so that no transient output change occurs.

Cycle—Change in Output State

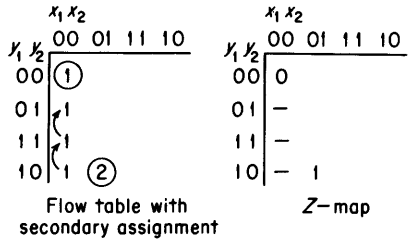


Figure 17-7

The output must change state only once, and therefore the choice of optional output states is restricted to the solutions in Fig. 17-8.

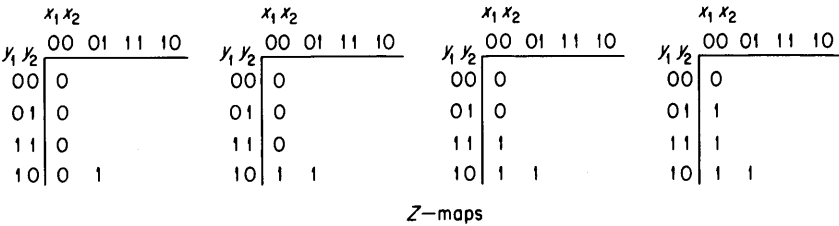


Figure 17-8

Race—No Change in Output State

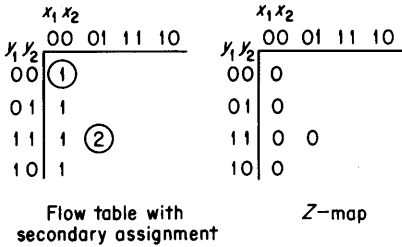


Figure 17-9

$Z = 0$ must be assigned to all unstable states so that no transient output change occurs.

Race—Change in Output State

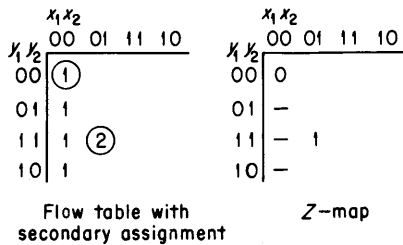


Figure 17-10

The output must change state only once, and therefore the choice of optional output states is restricted to the solutions in Fig. 17-11.

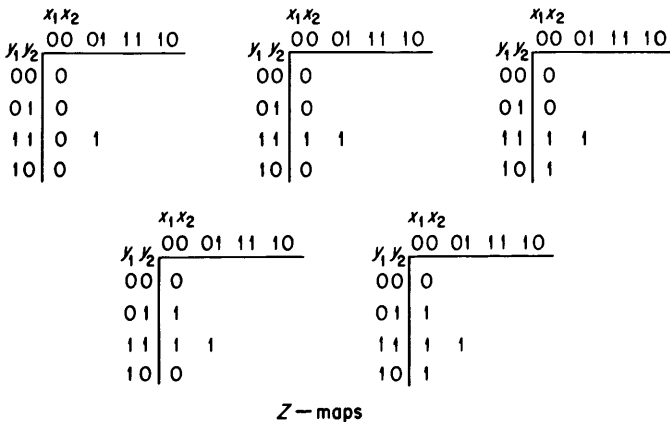


Figure 17-11

Transient Outputs; Cyclic Specifications

Transient outputs, associated only with particular *unstable* states, may sometimes be specified in a sequential circuit requirement. For example, in the flow table in Fig. 17-12, a transient output associated with unstable state 2 might be desired, the expression for this output being

$$Z = \bar{x}_1x_2\bar{y}_1\bar{y}_2$$

Cycles may be prescribed for the express purpose of introducing a series of transient outputs, as in the example in Fig. 17-13.

A continuous series of transient outputs is sometimes desired, as in the example in Fig. 17-14. When an input change to $x_1x_2 = 01$ occurs, the

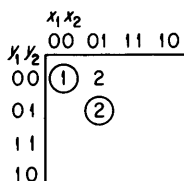


Figure 17-12

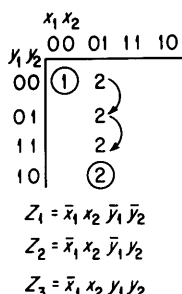


Figure 17-13

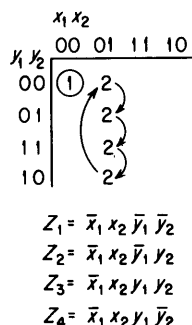


Figure 17-14

circuit will cycle continuously, producing the series of transient outputs until another input change occurs.

When a cycle is required as part of the original circuit specifications, such as in furnishing a series of transient outputs, transitions *from* and *to* the same rows of a flow table occur in the same column, imposing restrictions on the applicability of the secondary assignment patterns previously discussed. For example, in the left three columns of the flow table in Fig. 17-15, transitions between all six pairs of rows occur. As far as these left three columns are concerned, the secondary assignment (pattern #5) in Fig. 17-16 is satisfactory. However, this assignment is not satisfactory for the $x_1 x_2 = 10$ column. If the a to b transition is prescribed by the cycle $afgb$, the b to c transition must be prescribed by the cycle $bhca$, and there are then no spare rows adjacent to c to accomplish the c to d transition. If the a to b transition is prescribed by the alternative cycle $aejb$, the b to c transition must be prescribed by the cycle $bgec$, and again there are no spare rows adjacent to c to accomplish the c to d transition.

Alternate secondary assignments with the same pattern may be applicable when such a condition exists. For instance, the assignment in Fig. 17-17 is satisfactory.

Pattern #6 is unique in that it is applicable to any four-row flow table, regardless of any type of cycle that may be prescribed.

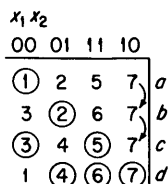


Figure 17-15

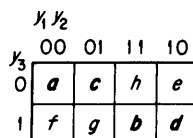


Figure 17-16

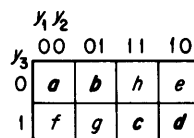


Figure 17-17

Hazards

The physical devices used to implement switching circuits are not ideal in the sense that the relationships

$$\text{if } X = 0, \text{ then } \bar{X} = 1$$

$$\text{if } X = 1, \text{ then } \bar{X} = 0$$

do not always exist. During transition times, the relationships

$$X = \bar{X} = 0$$

$$\text{or } X = \bar{X} = 1$$

may briefly exist.

Some examples of the consequences, in sequential circuits, of the imperfection in devices will now be studied. Implementation of the expression

$$AB + \bar{A}C$$

will be used as a running example.

EXAMPLE:

Consider the relay implementation in Fig. 17-18, in which the transfer contacts on relay A are of the *break-before-make* type.

Assume a condition of relays B and C operated, and relay A changing state. The circuit is closed before and after the change, but for a brief interval of time during the transition of relay A , both the A and \bar{A} contacts are open, and the circuit is therefore open. Such a false circuit condition is called a *hazard*.

Circuit hazards are undesirable not only because of the momentary false output conditions; if the hazard exists in a secondary excitation circuit, the more serious consequence of incorrect circuit operation can result.

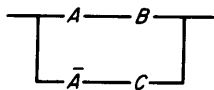


Figure 17-18

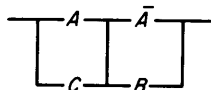


Figure 17-19

The hazard in the above circuit can be eliminated by making use of the relationship

$$AB + \bar{A}C = (A + C)(\bar{A} + B)$$

and implementing the circuit as in Fig. 17-19. With this implementation, when both the A and \bar{A} contacts are open during the transition of relay A ,

the circuit remains closed, a path being established through the closed B and C contacts.

EXAMPLE:

Now consider the relay implementation in 17-20 in which the transfer contacts on relay A are of the *make-before break* or continuity transfer type.

Assume a condition of relays B and C unoperated, and relay A changing state. The circuit is open before and after the change, but for a brief interval of time during the transition of relay A , both the A and \bar{A} contacts are closed, and the circuit is therefore closed.

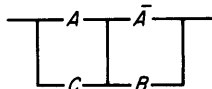


Figure 17-20

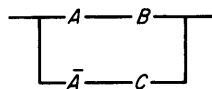


Figure 17-21

This hazard can be eliminated by implementing the circuit as in Fig. 17-21. With this implementation, when both the A and \bar{A} contacts are closed during the transition of relay A , the circuit remains open, the open B and C contacts preventing any path from being established.

The two types of hazards just discussed are illustrated in timing charts in Fig. 17-22. Note that the hazards actually exist only if the assumed conditions occur.

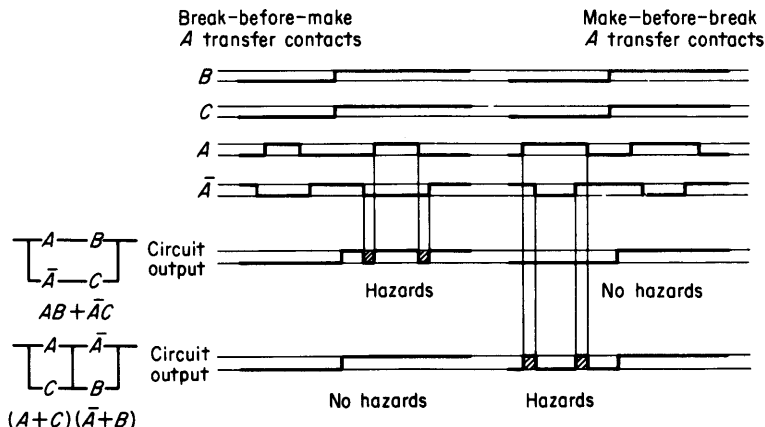


Figure 17-22

EXAMPLE:

Electronic implementation will now be considered, in which an inverter is used to implement \bar{A} (Fig. 17-23). There is an inherent delay between

a change at the inverter input and the corresponding change at the inverter output, as illustrated in the timing chart in Fig. 17-24

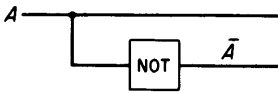


Figure 17-23

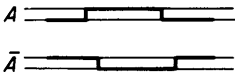


Figure 17-24

Implementation of either expression,

$$AB + \bar{A}C \text{ or } (A + C)(\bar{A} + B)$$

can result in one of the types of hazards previously discussed, as illustrated in the timing charts in Fig. 17-25.

Maps can be used in the identification and elimination of such possible hazards. The map associated with the preceding examples is shown in Fig. 17-26.

The expression $AB + \bar{A}C$ is obtained by grouping the 1-squares as in Fig. 17-27. A hazard can exist when a circuit change causes a movement between two states not in the same group; for example, between

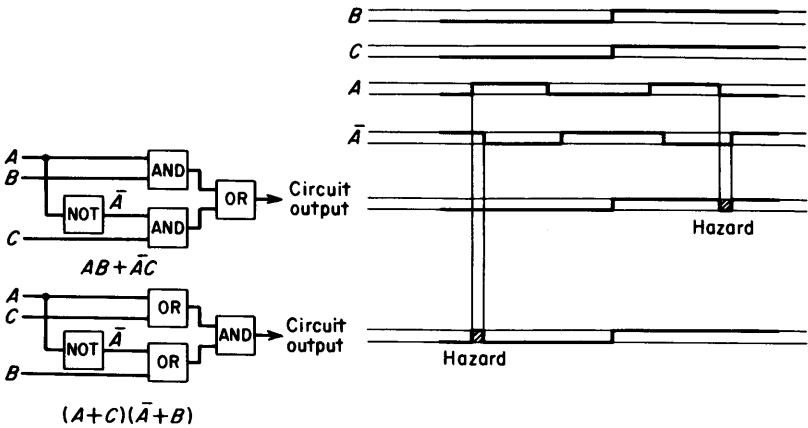


Figure 17-25

		AB			
		00	01	11	10
C	0	0	0	1	0
	1	1	1	1	0

Figure 17-26

		AB			
		00	01	11	10
C	0	0	0	1	0
	1	1	1	1	0

Figure 17-27

$ABC = 011$ and 111 , as indicated by the arrows in Fig. 17-27. The hazard can be eliminated by grouping the 1-squares between which this movement exists, as shown in the map and corresponding circuit in Fig. 17-28. The hazard is eliminated since the logic block corresponding to the term BC maintains the circuit output in the *on* state when $BC = 11$.

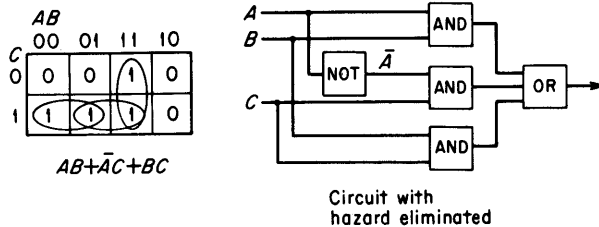


Figure 17-28

The expression $(A + C)(\bar{A} + B)$ is obtained by using the complementary approach and grouping the 0-squares on the map as in Fig. 17-29. A hazard can exist in this case, when there is a circuit change between $ABC = 000$ and 100 , as indicated by the arrows. The hazard can be eliminated by grouping these 0-squares, as shown in the map and corresponding circuit in Fig. 17-30. The hazard is eliminated since the logic block corresponding to the factor $(B + C)$ maintains the circuit output in the *off* state when $BC = 00$.

To eliminate hazards in sequential circuits, redundancy is thus sometimes required.

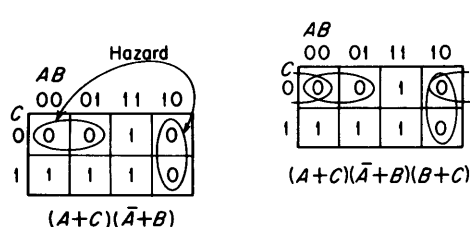


Figure 17-29

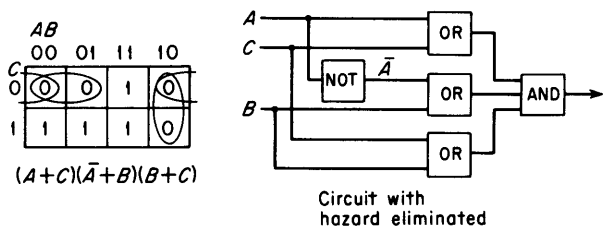


Figure 17-30

ANOTHER EXAMPLE:

In Fig. 17-31, if a circuit is implemented from the expression $\bar{C}\bar{D} + B\bar{C} + AD$, a hazard can exist when there is a circuit change between $ABCD = 1000$ and 1001 , as indicated by the arrows. The hazard can be eliminated by adding the term $A\bar{C}$ to the expression, and implementing $\bar{C}\bar{D} + B\bar{C} + AD + A\bar{C}$.

If the circuit is implemented from the expression $(A + B + \bar{D})(A + \bar{C})(\bar{C} + D)$, obtained by the complementary approach, Fig. 17-32, no hazard can exist.

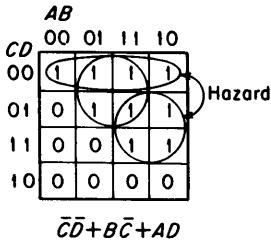


Figure 17-31

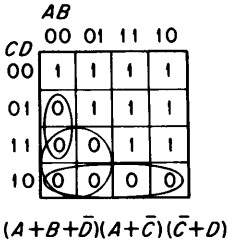


Figure 17-32

Hazards may occur in secondary excitation (Y) or output (Z) circuits. Before modifying a circuit to eliminate a possible hazard, it should be determined whether or not the corresponding condition can actually occur. This is done by reference to the merged flow table with secondary assignment, the primitive flow table, and the physical implementation. The merged flow table with secondary assignment may show that the hazard cannot exist because the associated circuit change can never occur. On the other hand, if the merged flow table indicates that the change may occur, the change is verified in the primitive flow table. The primitive flow table may show that the circuit change can never occur and that therefore the hazard cannot exist. However, if it is verified that the change can occur, the type of change—0 to 1, 1 to 0, or both—and the states of the other variables are correlated with the physical implementation (see, for example, Figs. 17-22 and 17-25) for the final determination of whether the condition can actually occur.

If the condition can never occur, then no hazard actually exists. If the condition can occur, then the hazard must be eliminated.

The literals A , B , C , and D , used in this section, were chosen for generality, and may represent either x 's or y 's.

The hazards discussed in this section are called static hazards. Other types of hazards are discussed in the literature (see Related Literature section).

Most-Economical Circuit Considerations

There are many factors involved in obtaining the most economical circuit. Although it is generally desirable to remove all redundant stable states, it occasionally is economically advantageous to retain or even purposely add redundant states. The choice of mergers can affect circuit economy,

as can the secondary assignment and the choice of optional transitions for the assignment.

As in the case of any multi-output circuit, the entire circuit must be evaluated as a whole, since the various secondary excitation and output circuits may be able to share logic blocks or contacts in common. When there are alternate ways of reading the maps, consideration should be given to the compatibility of expressions from an over-all circuit economy standpoint. The following example illustrates this point.

EXAMPLE:

$x_1 x_2$					
y	00	01	11	10	
0	0	1	0	0	
1	0	1	1	0	

$x_1 x_2$					
y	00	01	11	10	
0	0	-	0	0	
1	-	1	1	-	

Y -map		$Y = \bar{x}_1 x_2 + x_2 y$
		$= x_2 (\bar{x}_1 + y)$

Z -map		$Z = y$
----------	--	---------

Figure 17-33

The expressions above lead to the relay implementation shown in Fig. 17-34. The y contact in the output circuit can be eliminated, however, if the alternate output expression

$$\begin{aligned} Z &= \bar{x}_1 x_2 + x_2 y \\ &= x_2 (\bar{x}_1 + y) \end{aligned}$$

is used. This expression is identical to that for the secondary excitation, and the same circuit is thus made to serve two functions (Fig. 17-35).

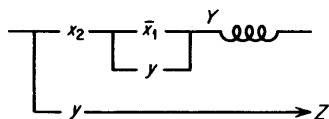


Figure 17-34

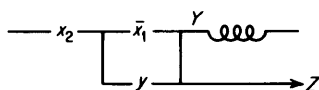


Figure 17-35

Illustrative Problem

The following problem, from word statement to final circuit (Figs. 17-36 through 17-40), reviews some of the principles discussed. An electronic sequential switching circuit is to have two inputs, x_1 and x_2 , and one output, Z . Z is to turn on when x_1 turns on; Z is to turn off when x_2 turns off. Only one input can change state at a time.

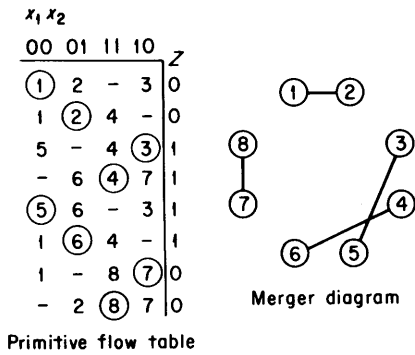


Figure 17-36

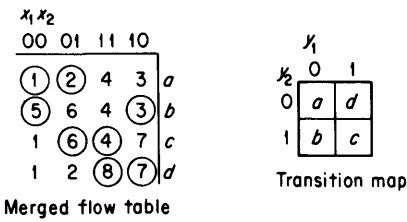


Figure 17-37

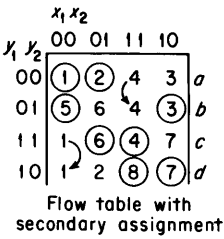
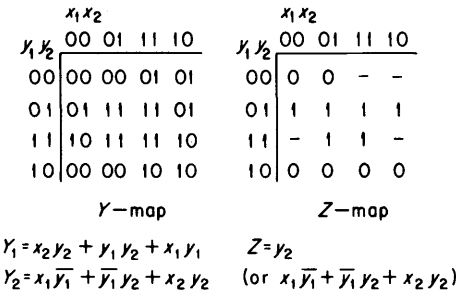
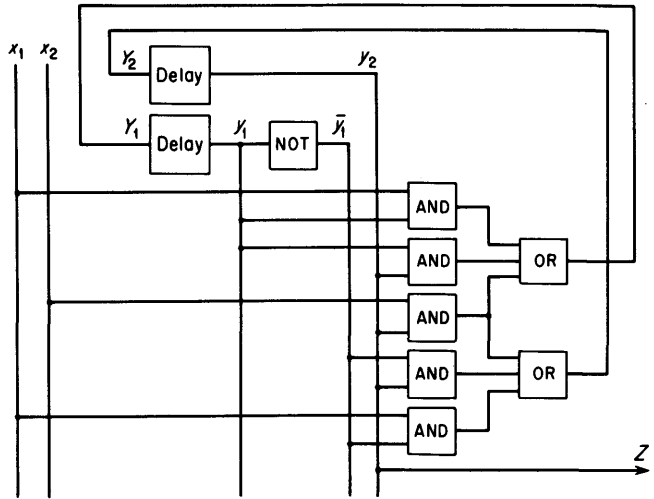


Figure 17-38





Circuit diagram

Figure 17-40

PROBLEMS

1. From the flow tables in Fig. 17-41, draw the Z-map and obtain the output expressions.

$x_1 x_2$					$Z_1 Z_2$	$x_1 x_2$					$y_1 y_2$
00	01	11	10			00	01	11	10		
①	2	—	3	10		①	2	⑤	3	00	
4	②	5	—	11		01	1	⑦	6	③	
1	—	6	③	01		11	④	7	⑥	8	
④	7	—	8	00		10	4	②	5	⑧	
—	2	⑤	3	01							
—	7	⑥	8	00							
1	⑦	6	—	11							
4	—	5	⑧	10							

Figure 17-41

- *2. Using another merger of the primitive flow table in Fig. 17-41, draw the Z-map and obtain the output expressions.
3. A sequential circuit is to have two inputs, x_1 and x_2 and one output, Z . The output is to be on only when $x_1 x_2 = 01$, or when $x_1 x_2 = 11$ imme-

diately following $x_1x_2 = 01$. The output state is optional for $x_1x_2 = 11$ immediately following $x_1x_2 = 00$. All input changes are possible. Design the circuit for electronic implementation.

- *4.** A sequential circuits is to have two inputs, x_1 and x_2 , and one output Z . Z is to turn on when x_2 turns on, provided that x_1 is on at the time. Z is to turn off when x_2 turns off. If x_1 changes state simultaneously with x_2 turning on, the circuit action is optional. All input changes are possible. Design the circuit for electronic implementation.

18

Pulse-Input Sequential Circuits I

In preceding chapters, electronic sequential circuits of the level-input type were discussed. In these level-input sequential circuits, the memory and delay properties were realized by feedback paths, and level-outputs were obtained. In this and the next chapter, pulse-input electronic sequential circuits will be discussed. In these pulse-input sequential circuits, the memory and delay properties are obtained with bistable electronic devices called *flip flops*, and pulse-outputs or level-outputs can be obtained.

In level-input sequential circuits, a change in circuit state is initiated by a change in input voltage level, that is, a change from “—” to “+,” or from “+” to “—.”

In pulse-input sequential circuits, a change in circuit state is initiated by an input *pulse*: a change from one voltage level to the other, followed by a return to the initial voltage level, the time during which the voltage deviates from the original level being of relatively brief duration compared with the time between deviations. For example, an input may normally be at the “—” level, change to the “+” level, and after a relatively brief interval

of time, return to the “−” level. This brief deviation to the “+” voltage level is called a *positive pulse*. A similar deviation from the “+” level to the “−” level is called a *negative pulse*.



Figure 18-1

Flip Flops

A flip flop is a “memory” device having two stable states which will be called “on” and “off.” A flip flop may have one, two, or three *pulse-inputs*¹, and has two complementary *level-outputs*, y and \bar{y} .

One of the flip flop outputs is at the “+” voltage level and the other is at the “−” voltage level at any time. When the y output is at the “+” voltage level, and the \bar{y} output is at the “−” level, the flip flop is said to be “on”; when y is “−” and \bar{y} is “+,” the flip flop is said to be “off.”

A flip flop responds to *pulses* at the inputs. A flip flop will remain in a given state until a proper input pulse is applied; that is, an input pulse causes a flip flop to change its state from “off” to “on” or from “on” to “off.”

Several types of flip flops and their applications in pulse-input sequential circuits will be discussed later. At this time, one common type of flip flop, the set-reset or *S-R* flip flop, will be described in some detail, and some modifications shown. A basic flip flop circuit is shown in Fig. 18-2. The

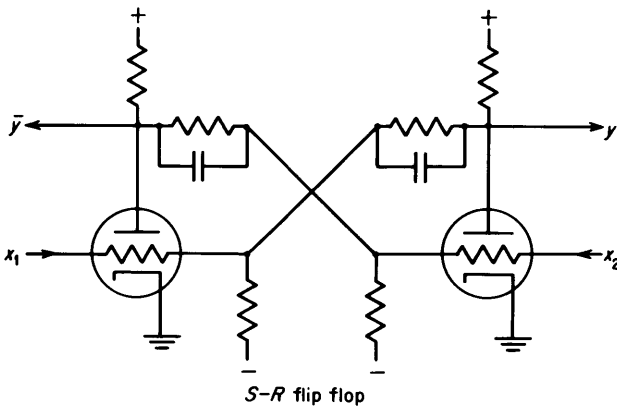


Figure 18-2

¹Flip flops respond also to changes in input level, but this mode of operation is not considered here.

inputs are x_1 and x_2 , and the outputs, y and \bar{y} . Vacuum tube implementation is illustrated; however, the flip flop can be similarly implemented using transistors.

At any given time, one triode is conducting and the other triode is not. When the left-hand triode is conducting, \bar{y} is at the lower or “—” voltage level and y is at the higher or “+” voltage level, and the flip flop is on. When the right-hand triode is conducting, y is at the “—” voltage level and \bar{y} is at the “+” voltage level, and the flip flop is off.

The flip flop will not change its state until a pulse of the proper polarity is applied to one of its inputs. For example, assume the flip flop to be on. The left-hand triode is conducting and its plate is at the lower voltage level. Since the plate of each triode is cross-coupled to the grid of the other triode, the lower voltage level at the plate of the left-hand triode is applied to the grid of the right-hand triode. The lower voltage level at the grid of the right-hand triode prevents this triode from conducting, and since it is not conducting, its plate is at the higher voltage level. The higher voltage level at the plate of the right-hand triode is applied through the cross-coupling to the grid of the left-hand triode, maintaining conduction in this triode. Thus, the flip flop is in a stable state.

When the flip flop is in the opposite stable state, off, the right-hand triode is conducting and the left-hand triode is not conducting.

Assume the flip flop to be on. To cause the flip flop to change state, from on to off, a negative pulse can be applied at the x_1 input, or a positive pulse can be applied at the x_2 input. Assume, for the sake of example, that a negative pulse is applied at x_1 . This pulse momentarily causes the grid of the left-hand triode to go negative, preventing conduction in this triode. The plate of the left-hand triode goes positive, and this positive voltage is applied to the grid of the right-hand triode, causing this triode to conduct. Conduction through the right-hand triode causes its plate to go negative, and this negative voltage is applied to the grid of the left-hand triode, preventing this triode from conducting even though the negative pulse at the x_1 input is no longer present. The pulse thus causes the flip flop to change its state from on to off.

If the flip flop is on, a positive pulse at the x_2 input causes the same circuit action, and the flip flop turns off. If the flip flop is on, a positive pulse at the x_1 input or a negative pulse at the x_2 input causes no change in the flip flop state, and the flip flop remains on. If the flip flop is off, a positive pulse at the x_1 input or a negative pulse at the x_2 input causes the flip flop to turn on. If the flip flop is off, a negative pulse at x_1 or a positive pulse at x_2 causes no change in the flip flop state, and the flip flop remains off.

A summary of the effects of positive and negative pulses at the two inputs with the flip flop initially in each state is given in the following table.

	Initial state of flip flop	
	ON	OFF
Negative pulse at x_1 or Positive pulse at x_2	Flip flop turns OFF	Flip flop stays OFF
Positive pulse at x_1 or Negative pulse at x_2	Flip flop stays ON	Flip flop turns ON

The symbol used for the *S-R* flip flop is shown in Fig. 18-3.

A pulse at the *S* (set) input causes the flip flop to turn on or stay on, depending upon its initial state. A pulse at the *R* (reset) input causes the flip flop to turn off or stay off. Which of the inputs in Fig. 18-2 would be labeled *S* and which would be labeled *R* depends upon the polarity of the input pulses: with positive input pulses, x_1 would be labeled *S*, and x_2 would be labeled *R*; with negative input pulses, x_2 would be labeled *S*, and x_1 would be labeled *R*.

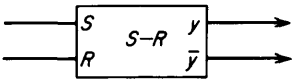


Figure 18-3

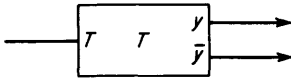


Figure 18-4

If, in the flip flop circuit shown in Fig. 18-2, both grids are connected through capacitors to a common input, a trigger or *T* flip flop results. An input pulse to a *T* flip flop causes the flip flop to change state: if the flip flop is initially on, the pulse will turn it off; if it is initially off, the pulse will turn it on. The symbol used for a *T* flip flop is shown in Fig. 18-4.

If, in the circuit shown in Fig. 18-2, *one* of the cross-coupling resistors is removed, a mono-stable device called a single-shot or one-shot multivibrator results. The triode with the grid cross-coupled only by the capacitor is the one normally conducting. If, say, a negative pulse is applied to the grid of this triode, the circuit will change state only temporarily, and then return to its original state. The length of time that the circuit remains in its unstable state is dependent upon the values of the circuit components. The single-shot multivibrator is useful for obtaining delays or for “pulse stretching.”

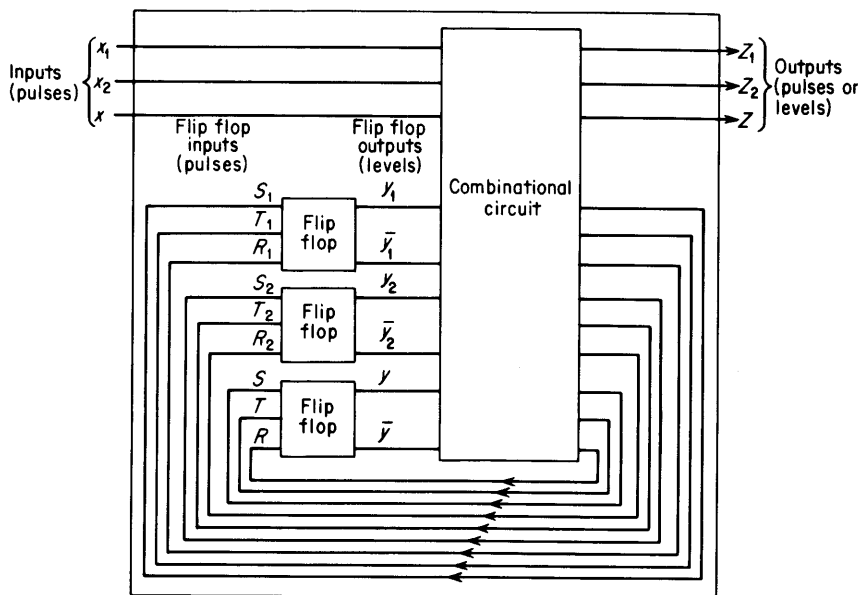
If *both* cross-coupling resistors are removed (and the grid resistors and the cathodes are placed at the same negative voltage), an unstable device called a multivibrator results. This circuit requires no inputs, the circuit continuously oscillating on and off as the triodes alternately conduct. The length of time that the circuit remains in each state is again dependent

upon the values of the circuit components. A multivibrator is useful as a pulse source.

Pulse-Input Sequential Circuits

A schematic diagram of a pulse-input sequential circuit is shown in Fig. 18-5. Note both the similarities and differences between this diagram and the analogous one for level-input sequential circuits shown in Fig. 13-1.

The sequential circuit inputs (x 's) are in the form of pulses. The flip flops are the secondaries. The flip flop inputs (S 's, T 's, and R 's) are in the form of pulses. Pulses at the flip flop inputs provide the secondary excitation. The flip flop outputs (y 's and \bar{y} 's) are in the form of levels. The states of the flip flop outputs are the secondary states.



Schematic diagram of pulse-input sequential switching circuit

Figure 18-5

The circuit input pulses may be directly applied to the flip flop inputs, or they may be switched with the flip flop outputs through combinational circuits. For example, a Boolean expression describing the circuitry for pulsing a flip flop input might be

$$S_1 = x_1 + x_2 \bar{y}_1 y_2$$

Since the flip flop inputs are in the form of pulses, every term in the corresponding flip flop excitation expression must contain an “x.”

The sequential circuit outputs (Z’s) may be in the form of pulses or levels. If pulse-outputs are specified, they are realized by switching together circuit input pulses and flip flop outputs. For example, an expression for a pulse-output might be

$$Z_1 = x_1 y_1 \bar{y}_2$$

If level-outputs are specified, they are realized by switching the flip flop outputs only. For example, an expression for a level-output might be

$$Z_1 = y_1 \bar{y}_2$$

It is assumed that the duration of the input pulses is relatively short compared with the response time (delay) of the flip flops. Thus, an input pulse “initiates” a change of state of a flip flop, and this change occurs *after* the pulse has “come and gone.” An input pulse to a flip flop may therefore be switched with an output on this same flip flop, the flip flop entering into its own control.

EXAMPLE:

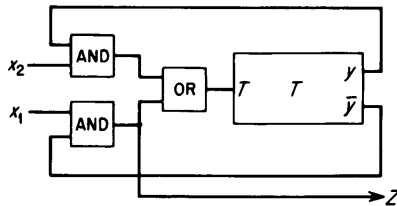


Figure 18-6

The relative durations of input pulses and flip flop response time may be depicted by the timing chart in Fig. 18-7.

When the flip flop is off ($\bar{y} = 1$), an x_1 input pulse, switched with the \bar{y} output, turns the flip flop on. When the flip flop is on ($y = 1$), an x_2 input pulse, switched with the y output, turns the flip flop off.

Note, by reference to the preceding circuit diagram and timing chart, that the output pulse Z is coincident with an x_1 input pulse occurring when the flip flop is off, even though this same pulse initiates the turning on of the flip flop.

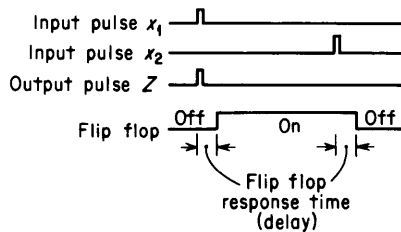


Figure 18-7

Because of the relative durations of input pulses and flip flop response time, two or more flip flops may simultaneously be caused to change state by the same input pulse without a concern about race conditions.

EXAMPLE:

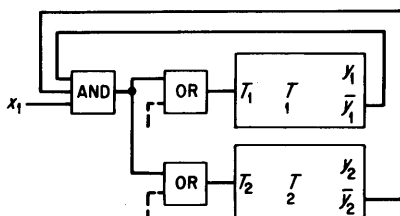


Figure 18-8

When both flip flops are off, an x_1 input pulse, switched with the \bar{y}_1 and \bar{y}_2 outputs, turns both flip flops on. Even if one flip flop responds faster than the other, no race condition exists. For example, if it is assumed that

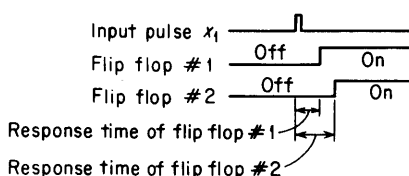


Figure 18-9

flip flop 1 responds faster than flip flop 2, the circuit action may be depicted by the timing chart in Fig. 18-9.

The problem of making secondary assignments to avoid critical races therefore does not exist for these pulse-input sequential circuits

as it did for level-input sequential circuits, and the secondary or flip flop assignments can be arbitrary (although one assignment may lead to a more economical circuit than another).

Another assumption that is made is that there is sufficient time between successive input pulses for the flip flops to complete their change of state. This time is governed by the slowest flip flop, that is, the flip flop with the longest response time.

In circuits in which the preceding assumptions do not hold true, that is, if a flip flop response time is shorter than the duration of an input pulse, or if successive input pulses can occur before a flip flop has completed its change of state, the problem of critical races may have to be considered, and a flip flop may not be able to enter into its own control.

Synthesis of Pulse-Input Sequential Circuits

The steps in the synthesis of pulse-input sequential circuits are summarized below.

- (1) The word statement of the problem is transformed into a flow table and usually also into a flow diagram. This flow table is different from the one used in the synthesis of level-input sequential circuits and rather more closely resembles the table used in the synthesis of reiterative circuits. The flow diagram contains the same information as the flow table, but permits a more graphic visualization of the entire circuit operation.
- (2) The flow table is tested for redundancy, and any redundant states can be eliminated.
- (3) A secondary assignment is made for the flow table, a combination of flip flop states being assigned to each circuit state.
- (4) Flip flop excitation maps are obtained from the flow table with secondary assignment. These maps differ from the usual maps with regard to the entries. The excitation expressions for pulsing the flip flop inputs are read from the maps. There are many types of flip flops, and for each type there is a specific set of rules for reading the maps.
- (5) The output expressions are read directly from the flow table with secondary assignment.
- (6) The sequential circuit is drawn from the excitation and output expressions.

Before discussing the construction of a flow diagram or flow table from a word statement of a problem, the diagrams and tables themselves will be described.

Flow Diagram

In a flow diagram, each circuit state is represented by a circled arbitrary number.

EXAMPLE:

All circuit states are stable, and transitions from state to state are effected by input pulses. Each of these transitions is represented on the diagram by an arrow labeled with the input pulse causing the transition; the arrow leaves the circled number representing the "initial" state and terminates in the circled number representing the "next" state.

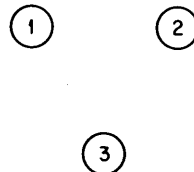


Figure 18-10

EXAMPLE:

The next state may be the same as the initial state; for example, in Fig.

18-11, if the circuit is in state 1 and an x_2 pulse occurs, the circuit remains in state 1; if the circuit is in state 3 and an x_1 pulse occurs, the circuit remains in state 3. Note that, in the flow diagram, there will be as many arrows *leaving* each circled number as there are input pulses (unless for an initial state it is impossible for certain input pulses to occur or the next state is optional).

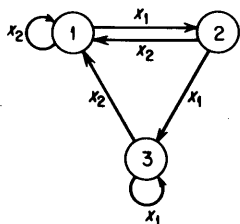


Figure 18-11

EXAMPLE:

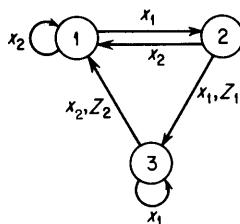


Figure 18-12

In the example, a Z_1 output pulse is coincident with an x_1 pulse occurring when the circuit is in state 2, and a Z_2 output pulse is coincident with an x_2 pulse occurring when the circuit is in state 3.

If the outputs are *levels*, they are associated with certain *circuit states*; on the flow diagram, level-outputs are labeled adjacent to the associated circuit state circled numbers.

EXAMPLE:

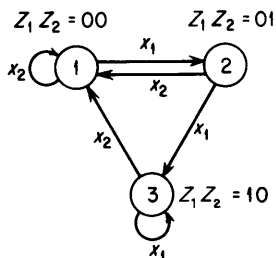


Figure 18-13

In Fig. 18-13, the Z_1 output is on when the circuit is in state 3, and the Z_2 output is on when the circuit is in state 2.

Flow Table

A flow table contains the same information as a flow diagram. There is a row in the table for each circuit state, and a column for each input pulse.

Each circuit state is assigned an arbitrary number which labels the corresponding row. Each column is labeled with an input pulse.

Each row label represents an initial state. Each “entry” in the table indicates the next state that will be reached when the circuit is initially in the state labeling that row and the input pulse labeling that column occurs.

Pulse-outputs are labeled adjacent to the entries corresponding to the associated transitions. Level-outputs are designated at the right of the rows labeled by the associated circuit states. Flow tables corresponding to the preceding two flow diagrams are shown in Fig. 18-14.

	x_1	x_2		x_1	x_2	$Z_1 Z_2$
1	2	1	1	2	1	00
2	3, Z_1	1	2	3	1	01
3	3	1, Z_2	3	3	1	10

Figure 18-14

Word Statement to Flow Diagram and Flow Table

In the construction of a flow diagram or flow table, it must be determined (a) how many circuit states are needed, and what is the associated information pertaining to past input sequences leading to each of these states, and (b) with the circuit in each of these states, to which state will a transition occur upon receipt of any input pulse.

(a), above, may be determined by the study of the problem statement, before starting the actual construction of the flow diagram or table, or it may be determined concurrently with (b), by starting the construction with an initial circuit state and adding additional states as required by the indicated transitions. The latter method is more commonly used.

The construction of a flow diagram and flow table from a word statement of the problem will now be illustrated by the use of some examples.

The circuit requirements in the first four examples involve two pulse-inputs, x_1 and x_2 , and one pulse-output, Z .

EXAMPLE § 1:

An output pulse Z is to be coincident with the first x_2 pulse immediately following an x_1 pulse.

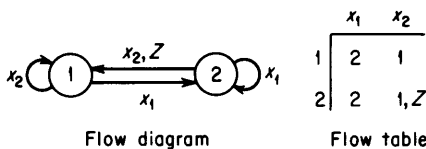


Figure 18-15

In this problem, it is sufficient to recognize two different conditions: the last input pulse that occurred was an x_1 pulse, or the last pulse was an x_2 pulse.

Thus, two circuit states are required, and they are assigned as follows:

1: the last input pulse was an x_2 pulse.

2: the last input pulse was an x_1 pulse.

An output pulse is associated with an x_2 input pulse occurring when the circuit is in state 2, since such an x_2 pulse will be the first immediately following an x_1 pulse.

It is assumed in this problem, and in all of the problems that follow, that circuit state 1 is the "power on" state, that is, when the power is first turned on, the circuit will be in state 1.

In this example, the circuit states were assigned as shown, rather than in the reverse order, so that if the very first pulse to occur is an x_2 pulse, no output pulse will occur, this x_2 pulse not being preceded by an x_1 pulse. If the assignments had been made in the reverse order, and the first pulse was an x_2 pulse, an output pulse would occur.

If the problem statement were revised to read, "An output pulse Z is to be coincident with the first of a sequence of consecutive x_2 pulses," the state assignments would be made in the reverse order (Fig. 18-16) so that if the first pulse were an x_2 pulse, an output pulse would occur.

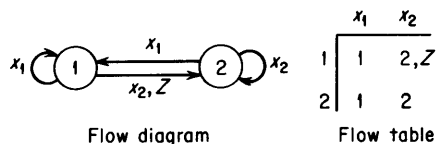


Figure 18-16

Once the circuit is "in operation," both assignments give the same circuit action; in other words, both problem statements are equivalent except for the output requirement of the very first input pulse if it is an x_2 pulse.

In pulse-input sequential circuit design, the assignment of circuit state 1 is frequently determined by the output requirements of an initial input sequence.

It is suggested that, for practice, the reader draw his own flow diagrams and flow tables for the examples that follow, and then compare his results with those given.

EXAMPLE § 2:

An output pulse Z is to be coincident with the second of a sequence of consecutive x_2 pulses immediately following an x_1 pulse.

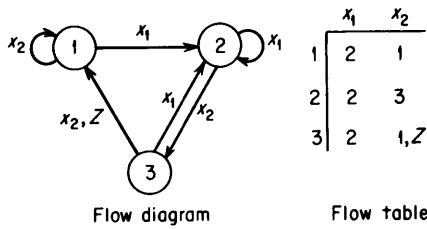


Figure 18-17

There are three conditions that must be recognized in this problem: the last input pulse was an x_1 pulse; the last input pulse was the first x_2 pulse immediately following an x_1 pulse; the last input pulse was an x_2 pulse other than the first immediately following an x_1 pulse.

Thus, three circuit states are required, and they are assigned as follows:

- 1: the last input pulse was an x_2 pulse other than the first immediately following an x_1 pulse.
- 2: the last input pulse was an x_1 pulse.
- 3: the last input pulse was the first x_2 pulse immediately following an x_1 pulse.

An output pulse is associated with an x_2 input pulse occurring when the circuit is in state 3, since such an x_2 pulse will be the second consecutive one immediately following an x_1 pulse.

Again, circuit state 1 was assigned so that if the first two input pulses were x_2 pulses, no output pulse would occur coincident with the second x_2 pulse. If the problem statement were revised to read, "An output pulse Z is to be coincident with the second of a sequence of consecutive x_2 pulses," the state assignments would be reordered as in Fig. 18-18.

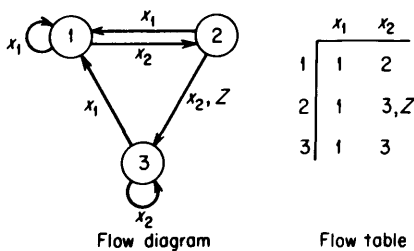


Figure 18-18

EXAMPLE § 3:

An output pulse Z is to be coincident with the first x_2 pulse immediately following two or more consecutive x_1 pulses.

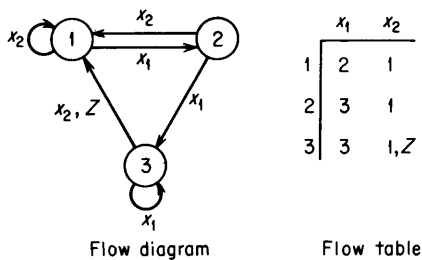


Figure 18-19

Three conditions must be recognized in this problem: the last input pulse was an x_2 pulse; the last input pulse was the first of a sequence of consecutive x_1 pulses; the last input pulse was the second or more of a sequence of consecutive x_1 pulses.

The three required circuit states are assigned as follows:

- 1: the last input pulse was an x_2 pulse.
- 2: the last input pulse was the first of a sequence of consecutive x_1 pulses.
- 3: the last input pulse was the second or more of a sequence of consecutive x_1 pulses.

An output pulse is associated with an x_2 input pulse occurring when the circuit is in state 3, since such an x_2 pulse will be the first one immediately following two or more consecutive x_1 pulses.

EXAMPLE § 4:

An output pulse Z is to be coincident with the first x_2 pulse immediately following exactly two consecutive x_1 pulses.

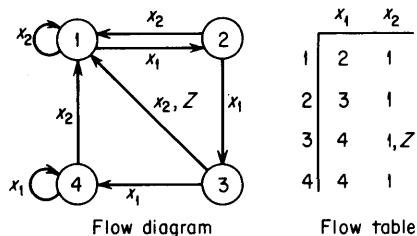


Figure 18-20

Four conditions must be recognized in this problem; these conditions and their assigned circuit states are:

- 1: the last input pulse was an x_2 pulse.
- 2: the last input pulse was the first of a sequence of consecutive x_1 pulses.
- 3: the last input pulse was the second of a sequence of consecutive x_1 pulses.
- 4: the last input pulse was the third or more of a sequence of consecutive x_1 pulses.

An output pulse is associated with an x_2 pulse occurring when the circuit is in state 3, since such an x_2 pulse will be the first one immediately following exactly two consecutive x_1 pulses.

The circuit requirements in the next two examples involve two pulse-inputs, x_1 and x_2 , and one level-output, Z .

EXAMPLE § 5:

If the output is off ($Z = 0$), it is to turn on ($Z = 1$) with an x_2 pulse. If the output is on, it is to turn off with the second of a sequence of consecutive x_1 pulses immediately following an x_2 pulse. No other input sequence is to cause any change in output.

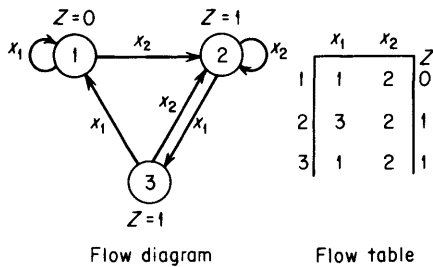


Figure 18-21

Three conditions must be recognized; these conditions and their assigned circuit states are:

- 1: $Z = 0$.
- 2: $Z = 1$. The last input pulse was an x_2 pulse.
- 3: $Z = 1$. The last input pulse was an x_1 pulse—the first following an x_2 pulse.

EXAMPLE § 6:

If the output is off, it is to turn on with the first of a sequence of x_2 pulses. If the output is on, it is to turn off with the second of a sequence of consecutive x_2 pulses. No other input sequence is to cause any change in output.

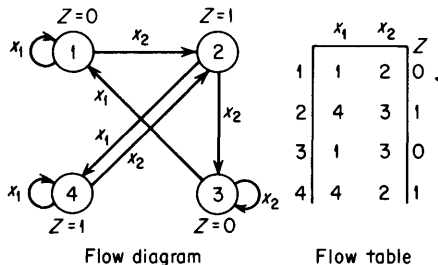


Figure 18-22

Four conditions that must be recognized, and their assigned circuit states, follow:

- 1: $Z = 0$. The last pulse was an x_1 pulse occurring when $Z = 0$.
- 2: $Z = 1$. The last pulse was an x_2 pulse—the first of a sequence of consecutive x_2 pulses.
- 3: $Z = 0$. The last pulse was an x_2 pulse—the second or more of a sequence of consecutive x_2 pulses.
- 4: $Z = 1$. The last pulse was an x_1 pulse occurring when $Z = 1$.

Many of these examples will be used in later discussions.

Elimination of Redundant States

As in level-input sequential circuits, redundancy may inadvertently be introduced in the design of pulse-input sequential circuits, redundant states being present in the flow diagram or flow table.

The concepts of equivalence and pseudo-equivalence in pulse-input sequential circuits are basically the same as those for level-input sequential circuits, and reference should be made to Chapter 14, in which the subject is covered more thoroughly. The concepts are restated briefly here.

Two circuit states are equivalent if:

- (1) The output conditions, pulse or level, associated with both states are the same, and
- (2) For each possible input pulse there is a transition from these states to the same or equivalent states.

If two states are equivalent, one of them is redundant and may be eliminated.

Following are shown some basic examples of equivalence. In all examples, only a portion of the flow table is shown, and in all examples, $1 \equiv 2$.

- (1)
- | | | | | |
|---|-------------|----------|---|-------------|
| | $x_1 \ x_2$ | | | $x_1 \ x_2$ |
| 1 | 4 3 | \equiv | 1 | 4 3 |
| 2 | 4 3 | | | |
-
- (2)
- | | | | | |
|---|-------------|----------|---|-------------|
| | $x_1 \ x_2$ | | | $x_1 \ x_2$ |
| 1 | 1 3, Z | \equiv | 1 | 1 3, Z |
| 2 | 1 3, Z | | | |
-
- (3)
- | | | | | |
|---|-------------|----------|---|-------------|
| | $x_1 \ x_2$ | | | $x_1 \ x_2$ |
| 1 | 2, Z 3 | \equiv | 1 | 1, Z 3 |
| 2 | 2, Z 3 | | | |

(4)

	x_1	x_2	Z	\equiv		x_1	x_2	Z
1	1	3	0		1	1	3	0
2	2	3	0					

(5)

	x_1	x_2	Z	\equiv		x_1	x_2	Z
1	2	3	1		1	1	3	1
2	1	3	1					

Following the practice of retaining the smaller-numbered circuit state, in all five examples every occurrence of a 2 is replaced by a 1. The two rows then become identical and are replaced by a single row.

The requirements for equivalence can also be stated in another way: two circuit states with the same output conditions can be made equivalent unless the equivalence depends upon a nonequivalence. (Note the interdependence of the the equivalence in the fourth and fifth examples.) An example with equivalence follows.

EXAMPLE:

A pulse-input sequential circuit is to have three inputs, x_1 , x_2 , and x_3 , and one pulse-output, Z . The output pulse is to be coincident with the first x_2 pulse immediately following either an x_1 pulse or an x_3 pulse.

A flow diagram and flow table for this circuit requirement are shown in Fig. 18-23.

Examination of the flow table shows that states 2 and 3 are equivalent since the output conditions associated with both states are the same, and for each input pulse there is a transition from these states to the same state.

The reduced flow diagram and flow table are shown in Fig. 18-24.

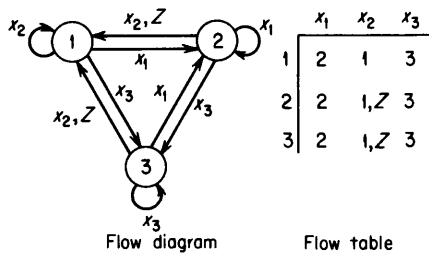


Figure 18-23

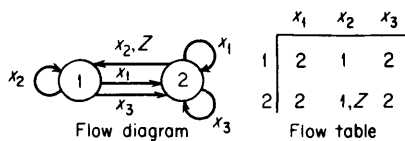


Figure 18-24

Two circuit states may be equivalent in all respects except for one or both of the following conditions:

- (1) For a given input pulse, there is a transition from one of these circuit states to a prescribed state, whereas the transition from the second circuit state is optional.
- (2) An output condition associated with one of these circuit states is prescribed, whereas for the second circuit state, the corresponding output condition is optional.

If either of the above conditions exists, the two circuit states are said to be pseudo-equivalent, and can be considered as equivalent.

Some basic examples follow. Again, only a portion of the flow table is shown and $1 \equiv 2$ in all cases.

(1) *Optional transition*

$$\begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 4, Z & 3 \end{array} \right. \\ 2 \left| \begin{array}{cc} 4, Z & - \end{array} \right. \end{array} \equiv \begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 4, Z & 3 \end{array} \right. \end{array}$$

(2) *Optional output*

$$\begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 4, Z & 3 \end{array} \right. \\ 2 \left| \begin{array}{cc} 4, - & 3 \end{array} \right. \end{array} \equiv \begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 4, Z & 3 \end{array} \right. \end{array}$$

(3) *Optional output*

$$\begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 4 & 3 \end{array} \right| \begin{array}{c} Z \\ 0 \end{array} \\ 2 \left| \begin{array}{cc} 4 & 3 \end{array} \right| \begin{array}{c} - \\ - \end{array} \end{array} \equiv \begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 4 & 3 \end{array} \right| \begin{array}{c} Z \\ 0 \end{array} \end{array}$$

In the next example, state reduction requires that a circuit state be made equivalent to two other circuit states which themselves are nonequivalent.

$$\begin{array}{c} x_1 \quad x_2 \\ 1 \left| \begin{array}{cc} 2 & 3 \end{array} \right| \begin{array}{c} Z \\ 0 \end{array} \\ 2 \left| \begin{array}{cc} 1 & 3 \end{array} \right| \begin{array}{c} 1 \\ 1 \end{array} \\ 3 \left| \begin{array}{cc} 3 & 2 \end{array} \right| \begin{array}{c} - \\ - \end{array} \end{array}$$

States 1 and 2 are nonequivalent. States 1 and 3 can be made equivalent if states 2 and 3 can be made equivalent. The equivalence of states 2 and 3, however, is dependent upon the equivalence of states 1 and 3. Therefore, the

equivalences $1 \equiv 3$ and $2 \equiv 3$ can be made, and the flow table reduces to

	x_1	x_2	Z
1	2	2	0
2	1	2	1

Secondary Assignment

In making a secondary assignment for a flow table, each circuit state is assigned some combination of flip flop states. Since race conditions are of no concern, the assignment can be arbitrary.

For two circuit states, only one flip flop is required, its “off” state being assigned to one circuit state, and its “on” state to the other. Two flip flops are required in circuits having three or four states; three flip flops are sufficient for up to eight circuit states; and so forth. In general, for m circuit states, n flip flops are required, where $2^n \geq m$.

Although the assignments are arbitrary, one assignment may lead to a more economical circuit than another. There is no need to try all possible assignments, however, since many of them are trivial variations of each other. For example, with four circuit states, there are twenty-four possible assignments but only three need be tried, there being eight trivial variations of each of the three. An arbitrary set of three nonequivalent assignments is shown in the following table.

	#1	#2	#3
	y_1y_2	y_1y_2	y_1y_2
1	00	00	00
2	01	01	11
3	11	10	01
4	10	11	10

By interchanging flip flops, an alternate for each of the three above assignments is obtained:

	y_1y_2	y_1y_2	y_1y_2
1	00	00	00
2	10	10	11
3	11	01	10
4	01	11	01

By interchanging the on and off states of one or both flip flops, three more variations can be obtained from each of the six assignments above, accounting for all possible (24) assignments.

The "all flip flops off" state, $y_1 y_2 = 00$, will arbitrarily be assigned to circuit state 1, the "power on" state, and furthermore, only the three assignments #1, #2, and #3 will be considered throughout the rest of this chapter.

Three examples of flow tables with secondary assignment are shown in Figs. 18-25 through 18-27. Note how these flow tables differ from those used in the synthesis of level-input sequential circuits. In the next section, these examples will be used to obtain flip flop excitation maps.

Note, in Fig. 18-27, that there are only three circuit states, and that flip flop state $y_1 y_2 = 10$ never occurs. Dashes are therefore entered in the $y_1 y_2 = 10$ row as optional entries.

	x_1	x_2		$y_1 y_2$	x_1	x_2
1	2	1	00	01	00	
2	3	1	01	11	00	
3	4	1, Z	11	10	00, Z	
4	4	1	10	10	00	

Flow table from example § 4 Flow table with secondary assignment #1

Figure 18-25

	x_1	x_2		$y_1 y_2$	x_1	x_2
1	2	1	00	11	00	
2	3	1	11	01	00	
3	4	1, Z	01	10	00, Z	
4	4	1	10	10	00	

Flow table from example § 4 Flow table with secondary assignment #3

Figure 18-26

	x_1	x_2	Z		$y_1 y_2$	x_1	x_2	Z
1	1	2	0	00	00	01	0	
2	3	2	1	01	11	01	1	
3	1	2	1	11	00	01	1	
				10	--	--	--	

Flow table from example § 5

	x_1	x_2	Z
1	1	2	0
2	3	2	1
3	1	2	1

Flow table with secondary assignment #1

Figure 18-27

In general, for an r -row flow table which requires n flip flops, where $2^n_{\min} \geq r$, there are $2^n!/r!(2^n - r)!$ ways of selecting r out of the 2^n possible combinations. For each of these ways, there are $r!$ permutations of assigning the r combinations to the r rows, making the total number of possible assignments:

$$\frac{2^n! r!}{r!(2^n - r)!}$$

For each of these assignments there are 2^n ways of interchanging the *on* and *off* states of the flip flops and there are $n!$ ways of interchanging flip flops. There are thus

$$\frac{2^n! r!}{r!(2^n - r)! 2^n \cdot n!} = \frac{(2^n - 1)!}{(2^n - r)! n!}$$

non-trivial assignments for an r -row flow table.

Some values are tabulated below.

r	n	Number of non-trivial assignments
2	1	1
3	2	3
4	2	3
5	3	140
6	3	420
7	3	840
8	3	840
9	4	10,810,800

PROBLEMS

- 1. Draw a flow diagram and flow table for the following circuit requirement: A sequential circuit is to have three pulse-inputs x_1 , x_2 and x_3 and two pulse-outputs Z_1 and Z_2 . The Z_1 pulse is to be coincident with the first x_2 pulse immediately following an x_1 pulse. The Z_2 pulse is to be coincident with all consecutive x_2 pulses immediately following an x_3 pulse.
- 2. Draw a flow diagram and flow table for the following circuit requirement: A sequential circuit is to have two pulse-inputs x_1 and x_2 and one pulse-output Z . The Z pulse is to be coincident with the second of two consecutive x_2 pulses immediately following exactly two consecutive x_1 pulses.

3. Draw a flow diagram and flow table for the following circuit requirement: A sequential circuit is to have two pulse-inputs x_1 and x_2 and one pulse-output Z . The Z pulse is to be coincident with the third and any further consecutive x_2 pulses immediately following exactly three consecutive x_1 pulses.
- *4. Draw a flow diagram and flow table for the following circuit requirement: A sequential circuit is to have two pulse-inputs x_1 and x_2 and one pulse-output Z . The Z pulse is to be coincident with the third consecutive x_2 pulse immediately following three or more consecutive x_1 pulses.

19

Pulse-Input Sequential Circuits II

Flip Flop Excitation Maps

In the next step of the procedure, flip flop excitation maps are drawn from the flow table with secondary assignment. The expressions for pulsing the flip flop inputs are read from these maps.

A map is drawn for each combination of circuit pulse-input and flip flop. For example, if there are two circuit inputs, say x_1 and x_2 , and three flip flops required, six maps are drawn. It is convenient to arrange the maps so that all maps corresponding to a particular flip flop are in the same row, and all maps corresponding to a particular circuit input are in the same column.

The outputs of all flip flops are variables for all maps. In addition, the circuit pulse-input defining a map is a variable for that map.

A set of maps for three flip flops FF_1 , FF_2 , and FF_3 , and two circuit pulse-inputs x_1 and x_2 is shown in Fig. 19-1.

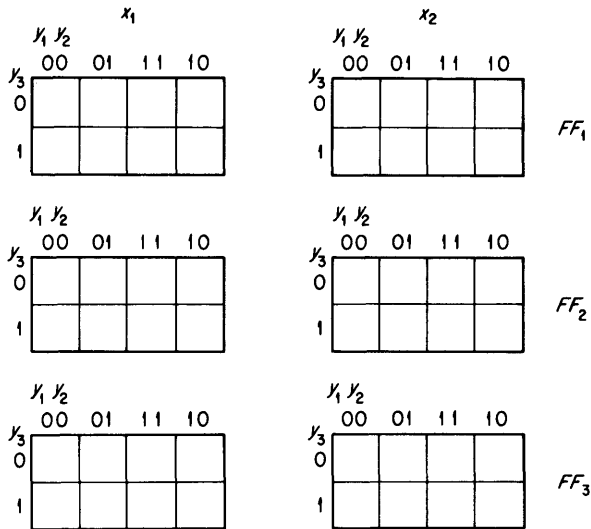


Figure 19-1

Map Entries

The flip flop excitation maps differ from the usual maps in that instead of there being only three possible entries, 1, 0, and —, there are five possible entries:

1, 0, 1, 0, —

The flip flop output states defining a particular map entry represent an *initial state*. The circuit pulse-input defining that map entry represents an input pulse occurring when the circuit is in the defined initial state.

Consider a map associated with a particular flip flop and pulse-input.

If the flip flop is *off* for an initial combination of flip flop output states, and following the input pulse, turns *on*, the corresponding map entry is a large one (**1**).

If the flip flop is initially *on*, and following the input pulse, turns *off*, the corresponding map entry is a large zero (**0**).

If the flip flop is initially *on*, and following the input pulse, remains *on*, the corresponding map entry is small a one (**1**).

If the flip flop is initially *off*, and following the input pulse, remains *off*, the corresponding map entry is a small zero (**0**).

An optional entry is indicated by a dash (—). An optional entry may

arise because a certain combination of flip flop states can never occur; or because for a certain initial flip flop state, a particular input pulse can never occur; or because for a certain initial flip flop state, we don't care what the circuit action is for a particular input pulse.

The map entries for each possible circuit action are summarized in the following table, the *off* state of a flip flop being represented by a 0, and the *on* state by a 1.

Circuit action	Map entry
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1
$0 \rightarrow 0$	0
Optional	—

$y_1 y_2$	x_1	x_2	Z
00	00	01	0
01	11	01	1
11	00	01	1
10	--	--	--

Flow table
from example § 5
Secondary assignment #1

Figure 19-2

As an example, flip flop excitation maps will be drawn from the flow table with secondary assignment in Fig. 19-2.

Refer to the first row of the flow table, in which the initial flip flop state is $y_1 y_2 = 00$. If an x_1 pulse occurs, the new flip flop state is $y_1 y_2 = 00$. The circuit action of flip flop 1, that is, the transition from the initial to the final output state, can be represented by $0 \rightarrow 0$, and the corresponding map entry is therefore 0. The circuit action of flip flop 2 is also $0 \rightarrow 0$, and the corresponding map entry is 0 (Fig. 19-3).

Still refer to the first row of the flow table, in which the initial flip flop state is $y_1 y_2 = 00$. If an x_2 pulse occurs, the new flip flop state is $y_1 y_2 = 01$. The circuit action of flip flop 1 is $0 \rightarrow 0$, and the map entry is 0. The circuit action of flip flop 2 is $0 \rightarrow 1$, and the map entry is **1** (Fig. 19-4).

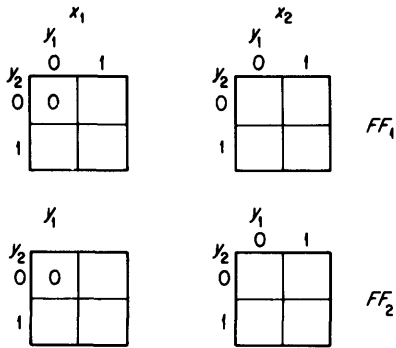


Figure 19-3

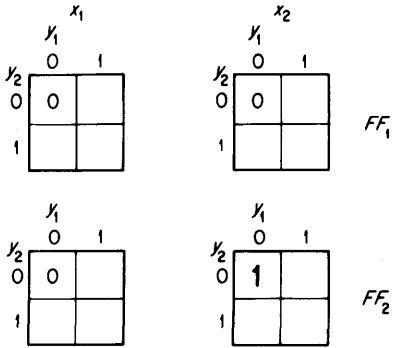


Figure 19-4

Now refer to the second row of the flow table, in which the initial flip flop state is $y_1y_2 = 01$. If an x_1 pulse occurs, the new flip flop state is $y_1y_2 = 11$. The circuit action of flip flop 1 is $0 \rightarrow 1$, and the map entry is **1**. The circuit action of flip flop 2 is $1 \rightarrow 1$, and the map entry is **1** (Fig. 19-5).

Still refer to the second row of the flow table, in which the initial flip flop state is $y_1y_2 = 01$. If an x_2 pulse occurs, the new flip flop state is $y_1y_2 = 01$. The circuit action of flip flop 1 is $0 \rightarrow 0$, and the map entry is **0**. The circuit action of flip flop 2 is $1 \rightarrow 1$, and the map entry is **1** (Fig. 19-6).

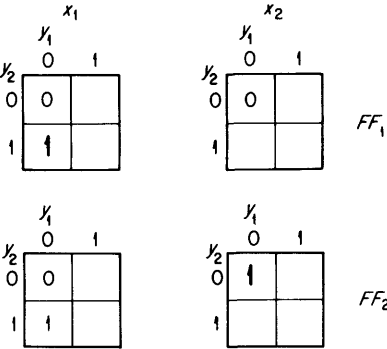


Figure 19-5

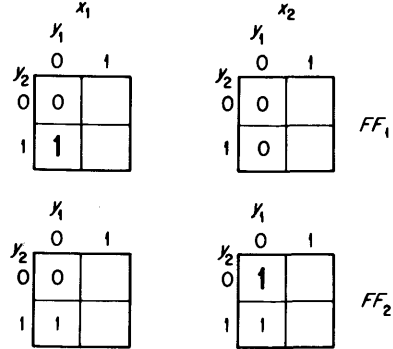


Figure 19-6

Refer now to the third row of the flow table, in which the initial flip flop state is $y_1y_2 = 11$. If an x_1 pulse occurs, the new flip flop state is $y_1y_2 = 00$. The circuit action of both flip flops is $1 \rightarrow 0$, and each corresponding map entry is **0** (Fig. 19-7).

Still refer to the third row of the flow table. If an x_2 pulse occurs when the initial flip flop state is $y_1y_2 = 11$, the new flip flop state is $y_1y_2 = 01$. The circuit action of flip flop 1 is $1 \rightarrow 0$, and the map entry is **0**. The circuit action of flip flop 2 is $1 \rightarrow 1$, and the map entry is **1**.

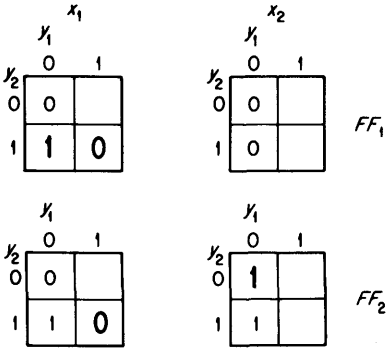


Figure 19-7

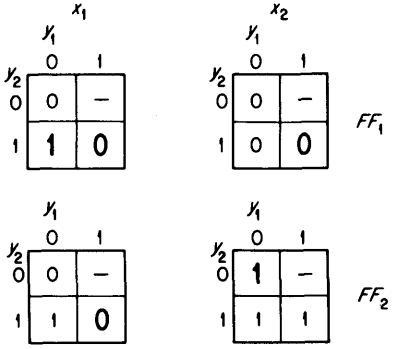


Figure 19-8

Since the $y_1y_2 = 10$ flip flop state is optional (refer to fourth row of flow table), all corresponding map entries are —'s. The completed flip flop excitation maps are shown in Fig. 19-8.

For practice, it is suggested that the reader draw flip flop excitation maps from the flow tables with secondary assignment in Figs. 19-9 and 19-10, and compare his results with the maps shown.

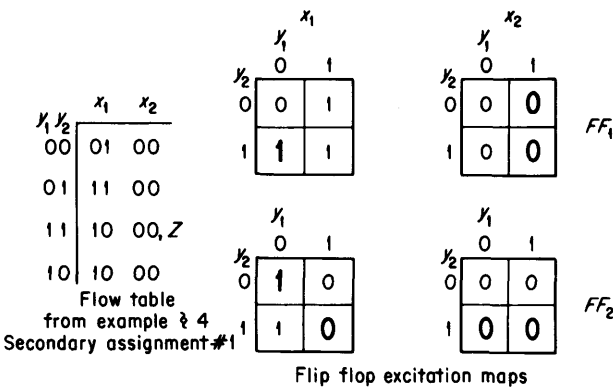


Figure 19-9

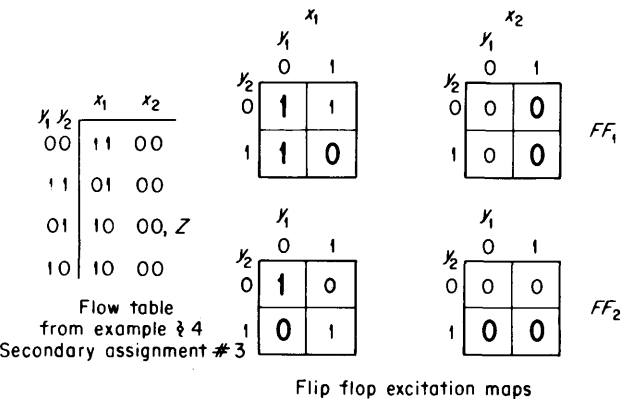


Figure 19-10

Note that with regard to flip excitation, the maps contain the same information as the corresponding flow table with secondary assignment. The information as contained in the maps is simply in a different and more useful form. The information pertaining to the outputs is retained in the flow table. The three flip flop excitation map sets obtained in this section will be used as examples in the next section.

Reading the Flip Flop Excitation Maps

The expressions for pulsing the flip flop inputs are read from the maps. There are many types of flip flops which will be discussed, and for each type there is a specific set of rules for reading the maps.

The advantage of the method presented here is that only one map set need be drawn. The flip flop excitation expressions for all types of flip flops can then be read from this map set merely by following the rules for each type of flip flop. In other methods, a map set must be drawn for each type of flip flop considered, each type having a specific set of rules for going from the flow table to the maps.

Five types of flip flops will be discussed: *S-R*, *S-R-SR*, *T*, *S-R-T*, and *S-R-SR-T*. Their operating characteristics will be analyzed, and from these characteristics their input pulse requirements will be obtained. From the input pulse requirements, their map-reading rules will be derived.

The method is general, and is adaptable to any other type of flip flop as well.

S-R Flip Flop

The *S-R* flip flop has two inputs: *S*(set) and *R*(reset). If the flip flop is off, a pulse on the *S* input will turn it on; a pulse on the *R* input will cause no change. If the flip flop is on, a pulse on the *R* input will turn it off; a pulse on the *S* input will cause no change. The *S* and *R* inputs of this flip flop must never be pulsed simultaneously, since the resulting circuit action is indeterminate. These operating characteristics are summarized in the following table. In the "Input" columns,

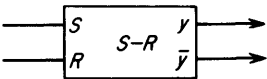


Figure 19-11

- 1 represents *input pulsed*
- 0 represents *input not pulsed*

Where no circuit action is indicated, the corresponding input pulsing is not allowable.

Input		Circuit action	
<i>S</i>	<i>R</i>		
0	0	$0 \rightarrow 0$	$1 \rightarrow 1$
1	0	$0 \rightarrow 1$	$1 \rightarrow 1$
0	1	$0 \rightarrow 0$	$1 \rightarrow 0$
1	1		

S-R flip flop
Operating characteristics

Based on the preceding operating characteristics, the *S-R* input pulse requirements for each possible circuit action are obtained. The following notation for input pulse requirements is used:

Input must be pulsed: 1
 Input must not be pulsed: 0
 Input may or may not be pulsed: —

Circuit action: 0 → 1

Map entry: 1

The *S* input must be pulsed.

The *R* input must not be pulsed.

<i>S</i>	<i>R</i>
1	0

Circuit action: 1 → 0

Map entry: 0

The *R* input must be pulsed.

The *S* input must not be pulsed.

<i>S</i>	<i>R</i>
0	1

Circuit action: 1 → 1

Map entry: 1

The *S* input may or may not be pulsed.

The *R* input must not be pulsed.

<i>S</i>	<i>R</i>
—	0

Circuit action: 0 → 0

Map entry: 0

The *R* input may or may not be pulsed.

The *S* input must not be pulsed.

<i>S</i>	<i>R</i>
0	—

Circuit action: Optional

Map entry: —

Any input may or may not be pulsed.

<i>S</i>	<i>R</i>
—	—

The *S-R* input pulse requirements are summarized in the following table.

Circuit action	Map entry	Input	
		<i>S</i>	<i>R</i>
$0 \rightarrow 1$	1	1	0
$1 \rightarrow 0$	0	0	1
$1 \rightarrow 1$	1	—	0
$0 \rightarrow 0$	0	0	—
Optional	—	—	—

S-R flip flop
Input pulse requirements

The map-reading rules for the *S-R* flip flop can be read from this table:
 Every **1** must be accounted for in the expression for pulsing the *S* input.
 Every **0** must be accounted for in the expression for pulsing the *R* input.
 Any 1 or — may be used optionally in the *S* input expression.
 Any 0 or — may be used optionally in the *R* input expression.

In the examples that follow, it is suggested that, for practice, the reader obtain the expressions for pulsing the flip flop inputs, and compare his results with those given.

EXAMPLE:

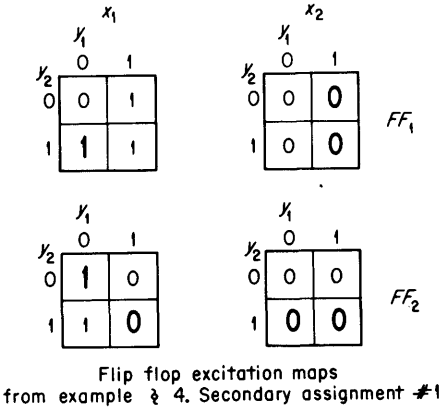


Figure 19-12

Reading the FF_1 maps for an *S-R* flip flop, the expression for pulsing the *S* input is:

$$S_1 = x_1 y_2$$

and the expression for pulsing the R input is:

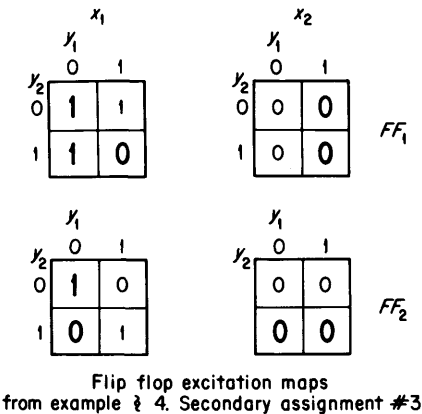
$$R_1 = x_2$$

Reading the FF_2 maps for an S - R flip flop,

$$S_2 = x_1\bar{y}_1$$

$$R_2 = x_1y_1 + x_2$$

EXAMPLE:



Flip flop excitation maps from example 4. Secondary assignment #3

Figure 19-13

S - R flip flops:

$$S_1 = x_1\bar{y}_1$$

$$R_1 = x_1y_1y_2 + x_2$$

$$S_2 = x_1\bar{y}_1\bar{y}_2$$

$$R_2 = x_1\bar{y}_1y_2 + x_2$$

S-R-SR Flip Flop

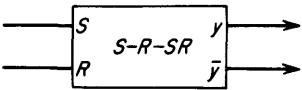


Figure 19-14

The S - R - SR flip flop has the same operating characteristics as the S - R flip flop with one exception: the S and R inputs may be pulsed simultaneously, in which case the flip flop will change state. These operating characteristics are summarized in the following table.

Input		Circuit action	
<i>S</i>	<i>R</i>		
0	0	$0 \rightarrow 0$	$1 \rightarrow 1$
1	0	$0 \rightarrow 1$	$1 \rightarrow 1$
0	1	$0 \rightarrow 0$	$1 \rightarrow 0$
1	1	$0 \rightarrow 1$	$1 \rightarrow 0$

S-R-SR flip flop
Operating characteristics

The *S-R-SR* input pulse requirements are:

Circuit action: $0 \rightarrow 1$

Map entry: **1**

The *S* input must be pulsed.

The *R* input may or may not be pulsed.

<i>S</i>	<i>R</i>
1	—

Circuit action: $1 \rightarrow 0$

Map entry: **0**

The *R* input must be pulsed.

The *S* input may or may not be pulsed.

<i>S</i>	<i>R</i>
—	1

Circuit action: $1 \rightarrow 1$

Map entry: **1**

The *S* input may or may not be pulsed.

The *R* input must not be pulsed.

<i>S</i>	<i>R</i>
—	0

Circuit action: $0 \rightarrow 0$

Map entry: **0**

The *R* input may or may not be pulsed.

The *S* input must not be pulsed.

<i>S</i>	<i>R</i>
0	—

Circuit action: Optional

Map entry: **—**

Any input may or may not be pulsed.

<i>S</i>	<i>R</i>
—	—

The *S-R-SR* input pulse requirements are summarized in the following table.

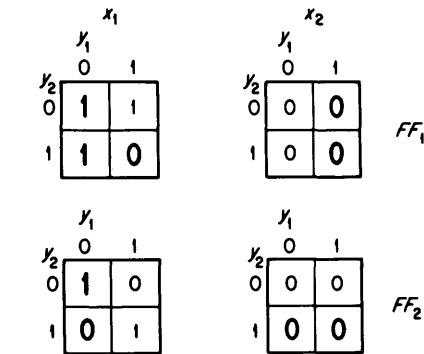
Circuit action	Map entry	Input	
		<i>S</i>	<i>R</i>
$0 \rightarrow 1$	1	1	—
$1 \rightarrow 0$	0	—	1
$1 \rightarrow 1$	1	—	0
$0 \rightarrow 0$	0	0	—
Optional	—	—	—

S-R-SR flip flop
Input pulse requirements

The map-reading rules for the *S-R-SR* flip flop can be read from this table:

- Every **1** must be accounted for in the *S* input expression.
- Every **0** must be accounted for in the *R* input expression.
- Any **0**, 1, or — may be used optionally in the *S* input expression.
- Any **1**, 0, or — may be used optionally in the *R* input expression.

EXAMPLE:



Flip flop excitation maps
from example § 4. Secondary assignment #3

Figure 19-15

S-R-SR flip flops:

$$S_1 = x_1$$

$$R_1 = x_1 y_2 + x_2$$

$$S_2 = x_1 \bar{y}_1$$

$$R_2 = x_1 \bar{y}_1 + x_2$$

EXAMPLE:

		x_1				x_2		
		y_1		y_1		y_1		
		0	1	0	1	0	1	
y_2	0	0	—	0	—	0	—	FF_1
	1	1	0	0	0	0	0	
		y_1		y_1		y_1		
		0	1	0	1	0	1	
y_2	0	0	—	1	—	1	—	FF_2
	1	1	0	1	1	1	1	

Flip flop excitation maps
from example § 5. Secondary assignment #1

Figure 19-16

S-R-SR flip flops:

$$S_1 = x_1 y_2$$

$$R_1 = x_1 + x_2$$

$$S_2 = x_2$$

$$R_2 = x_1 y_1$$

T Flip Flop

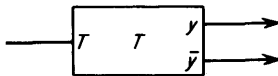


Figure 19-17

The T flip flop has one input: T (trigger). If the flip flop is off, a pulse on the T input will turn it on. If the flip flop is on, a pulse on the T input will turn it off. These operating characteristics are summarized in the following table.

Input	Circuit action	
T		
0	$0 \rightarrow 0$	$1 \rightarrow 1$
1	$0 \rightarrow 1$	$1 \rightarrow 0$

T flip flop
Operating characteristics

The T input pulse requirements are:

Circuit action: $0 \rightarrow 1$
The T input must be pulsed.

Map entry: **1**

$$\frac{T}{1}$$

Circuit action: $1 \rightarrow 0$
The T input must be pulsed.

Map entry: **0**

$$\frac{T}{1}$$

Circuit action: $1 \rightarrow 1$
The T input must not be pulsed.

Map entry: **1**

$$\frac{T}{0}$$

Circuit action: $0 \rightarrow 0$
The T input must not be pulsed.

Map entry: **0**

$$\frac{T}{0}$$

Circuit action: Optional
The T input may or may not be pulsed.

Map entry: **—**

$$\frac{T}{—}$$

The T input pulse requirements are summarized in the following table.

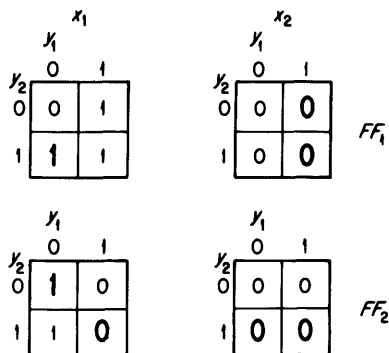
Circuit action	Map entry	Input
		T
$0 \rightarrow 1$	1	1
$1 \rightarrow 0$	0	1
$1 \rightarrow 1$	1	0
$0 \rightarrow 0$	0	0
Optional	—	—

T flip flop
Input pulse requirements

The map-reading rules for the T flip flop can be read from this table:

Every **1** and **0** must be accounted for in the T input expression.
Any — may be used optionally in the T input expression.

EXAMPLE:



Flip flop excitation maps
from example § 4. Secondary assignment #1

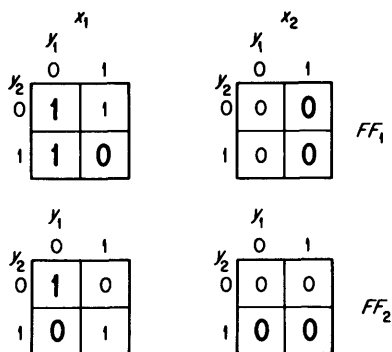
Figure 19-18

T flip flops:

$$T_1 = x_1 \bar{y}_1 y_2 + x_2 y_1$$

$$T_2 = x_1 \bar{y}_1 \bar{y}_2 + x_1 y_1 y_2 + x_2 y_2$$

EXAMPLE:



Flip flop excitation maps
from example § 4. Secondary assignment #3

Figure 19-19

T flip flops:

$$T_1 = x_1\bar{y}_1 + x_1y_2 + x_2y_1$$
$$T_2 = x_1\bar{y}_1 + x_2y_2$$

Note that the term $x_1\bar{y}_1$ need be implemented only once, this implementation being used in both the T_1 and T_2 circuits.

S-R-T Flip Flop

The *S-R-T* flip flop has three inputs *S*, *R*, and *T*, and has the combined operating characteristics of the *S-R* flip flop and the *T* flip flop. If the flip flop is off, a pulse on the *S* input or *T* input or both will turn it on; a pulse on the *R* input will cause no change. If the flip flop is on, a pulse on the *R* input or *T* input or both will turn it off; a pulse on the *S* input will cause no change.

The *S* and *R* inputs of this flip flop must never be pulsed simultaneously, since the resulting circuit action is indeterminate. For the same reason, the *S* and *T* inputs must never be pulsed simultaneously when the flip flop is on, and the *R* and *T* inputs must never be pulsed simultaneously when the flip flop is off.

These operating characteristics are summarized in the following table.

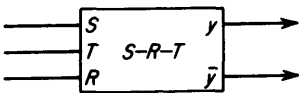


Figure 19-20

Input			Circuit action	
<i>S</i>	<i>R</i>	<i>T</i>		
0	0	0	0 → 0	1 → 1
1	0	0	0 → 1	1 → 1
0	1	0	0 → 0	1 → 0
0	0	1	0 → 1	1 → 0
1	0	1	0 → 1	
0	1	1		1 → 0
1	1	0		
1	1	1		

S-R-T flip flop
Operating characteristics

The *S-R-T* input pulse requirements are:

Circuit action: $0 \rightarrow 1$ Map entry: **1**Either the S or T input must be pulsed; the other may or may not be pulsed.The R input must not be pulsed.

S	R	T
1	0	—
—		1

Circuit action: $1 \rightarrow 0$ Map entry: **0**Either the R or T input must be pulsed; the other may or may not be pulsed.The S input must not be pulsed.

S	R	T
0	1	—
	—	1

Circuit action: $1 \rightarrow 1$ Map entry: **1**The S input may or may not be pulsed.The R input must not be pulsed.The T input must not be pulsed.

S	R	T
—	0	0

Circuit action: $0 \rightarrow 0$ Map entry: **0**The R input may or may not be pulsed.The S input must not be pulsed.The T input must not be pulsed.

S	R	T
0	—	0

Circuit action: Optional

Map entry: —

Any input may or may not be pulsed.

S	R	T
—	—	—

The S - R - T input pulse requirements are summarized in the following table.

Circuit action	Map entry	Input		
		<i>S</i>	<i>R</i>	<i>T</i>
0 → 1	1	1	0	—
		—		1
1 → 0	0	0	1	—
			—	1
1 → 1	1	—	0	0
0 → 0	0	0	—	0
Optional	—	—	—	—

S-R-T flip flop
Input pulse requirements

The map-reading rules for the *S-R-T* flip flop can be read from this table:

Every **1** must be accounted for either in the *S* input expression or the *T* input expression.

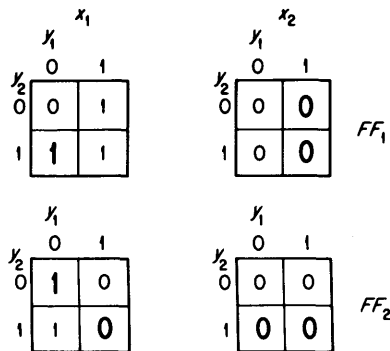
Every **0** must be accounted for either in the *R* input expression or the *T* input expression.

Any **1** accounted for in the *T* input expression, or any 1 or — may be used optionally in the *S* input expression.

Any **0** accounted for in the *T* input expression, or any 0 or — may be used optionally in the *R* input expression.

Any **1** accounted for in the *S* input expression, any **0** accounted for in the *R* input expression, or any — may be used optionally in the *T* input expression.

EXAMPLE:



Flip flop excitation maps
from example 4. Secondary assignment #1

Figure 19-21

S-R-T flip flops:

$$S_1 = x_1 y_2$$

$$R_1 = x_2$$

$$T_1 = (\text{unused})$$

$$S_2 = x_1 \bar{y}_1$$

$$R_2 = x_1 y_1 + x_2$$

$$T_2 = (\text{unused})$$

$$S_2 = x_1 \bar{y}_1$$

$$R_2 = x_2$$

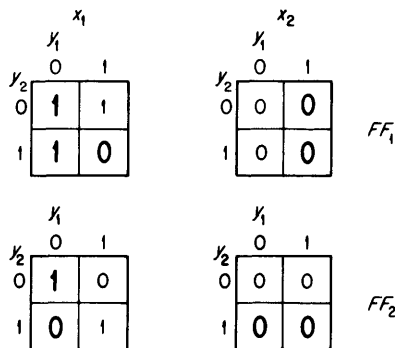
$$T_2 = x_1 y_1 y_2$$

The *S-R-T* flip flop has the characteristic that it can be turned on by pulsing the *S* or *T* input, and it can be turned off by pulsing the *R* or *T* input. Often, advantage can be taken of this "built-in" OR characteristic to achieve greater circuit economy.

An example of this is illustrated in the excitation of FF_2 above. The first solution can be implemented with two AND circuits and one OR circuit. The second solution may be more economically implemented with two AND circuits, the inherent OR characteristic of the *S-R-T* flip flop being utilized.

As illustrated in this example, it may sometimes be desirable to use larger terms in order to take advantage of the built-in OR characteristic: note the term $x_1 y_1$ in the first solution, versus the term $x_1 y_1 y_2$ in the second solution.

EXAMPLE:



Flip flop excitation maps
from example § 4. Secondary assignment #3

Figure 19-22

S-R-T flip flops:

$$\begin{aligned} S_1 &= x_1 \bar{y}_1 \\ R_1 &= x_2 \\ T_1 &= x_1 y_2 \\ S_2 &= (\text{unused}) \\ R_2 &= x_2 \\ T_2 &= x_1 \bar{y}_1 \end{aligned}$$

or

$$\begin{aligned} S_1 &= x_1 \bar{y}_2 \\ R_1 &= x_2 \\ T_1 &= x_1 y_2 \end{aligned}$$

The first solution for the excitation of FF_1 leads to a more economical circuit, since the term $x_1 \bar{y}_1$ need be implemented only once, this implementation being used in the S_1 and T_2 circuits.

S-R-SR-T Flip Flop

The *S-R-SR-T* flip flop has the same operating characteristics as the *S-R-T* flip flop with one exception: the *S* and *R* inputs may be pulsed simultaneously, in which case the flip flop will change state. The *S-R-SR-T* flip flop thus has the combined operating characteristics of the *S-R-SR* flip flop and the *T* flip flop.

When the flip flop is on, the *S* and *T* inputs must never be pulsed simultaneously unless the *R* input is also pulsed. When the flip flop is off, the *R* and *T* inputs must never be pulsed simultaneously unless the *S* input is also pulsed.

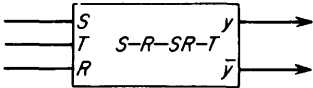


Figure 19-23

These operating characteristics are summarized in the following table.

Input			Circuit action	
S	R	T		
0	0	0	$0 \rightarrow 0$	$1 \rightarrow 1$
1	0	0	$0 \rightarrow 1$	$1 \rightarrow 1$
0	1	0	$0 \rightarrow 0$	$1 \rightarrow 0$
0	0	1	$0 \rightarrow 1$	$1 \rightarrow 0$
1	0	1	$0 \rightarrow 1$	
0	1	1		$1 \rightarrow 0$
1	1	0	$0 \rightarrow 1$	$1 \rightarrow 0$
1	1	1	$0 \rightarrow 1$	$1 \rightarrow 0$

S-R-SR-T flip flop
Operating characteristics

The S - R - SR - T input pulse requirements are:

Circuit action: $0 \rightarrow 1$

Map entry: **1**

Either the S or T input must be pulsed; the other may or may not be pulsed.

If the S input is pulsed, the R input may or may not be pulsed;

otherwise, the R input must not be pulsed.

S	R	T
1	—	—
—	0	1

Circuit action: $1 \rightarrow 1$

Map entry: **0**

Either the R or T input must be pulsed; the other may or may not be pulsed.

If the R input is pulsed, the S input may or may not be pulsed;

otherwise, the S input must not be pulsed.

S	R	T
—	1	—
0	—	1

Circuit action: $1 \rightarrow 1$

Map entry: **1**

The S input may or may not be pulsed.

The R input must not be pulsed.

The T input must not be pulsed.

S	R	T
—	0	0

Circuit action: $0 \rightarrow 0$

Map entry: **0**

The R input may or may not be pulsed.

The S input must not be pulsed.

The T input must not be pulsed.

S	R	T
0	—	0

Circuit action: Optional

Map entry: —

Any input may or may not be pulsed.

<i>S</i>	<i>R</i>	<i>T</i>
—	—	—

The *S-R-SR-T* input pulse requirements are summarized in the following table.

Circuit action	Map entry	Input		
		<i>S</i>	<i>R</i>	<i>T</i>
0 → 1	1	1	—	—
		—	0	1
1 → 0	0	—	1	—
		0	—	1
1 → 1	1	—	0	0
0 → 0	0	0	—	0
Optional	—	—	—	—

S-R-SR-T flip flop
Input pulse requirements

The map-reading rules for the *S-R-SR-T* flip flop can be read from this table.

- Every **1** must be accounted for either in the *S* input expression or the *T* input expression.
- Every **0** must be accounted for either in the *R* input expression or the *T* input expression.
- Any **1** accounted for in the *T* input expression, any **0** accounted for in the *R* input expression, or any 1 or — may be used optionally in the *S* input expression.
- Any **0** accounted for in the *T* input expression, any **1** accounted for in the *S* input expression, or any 0 or — may be used optionally in the *R* input expression.
- Any **1** accounted for in the *S* input expression, any **0** accounted for in the *R* input expression, or any — may be used optionally in the *T* input expression.

The input pulse requirements for the five flip flops discussed are summarized in the following table.

Circuit action	Map entry	Flip flop input pulse requirements										
		<i>S-R</i>		<i>S-R-SR</i>		<i>T</i>	<i>S-R-T</i>			<i>S-R-SR-T</i>		
		<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>T</i>	<i>S</i>	<i>R</i>	<i>T</i>	<i>S</i>	<i>R</i>	<i>T</i>
$0 \rightarrow 1$	1	1	0	1	—	1	1 —	0	— 1	1 —	— 0	— 1
$1 \rightarrow 0$	0	0	1	—	1	1	0 —	1 1	— 1	— 0	1 —	— 1
$1 \rightarrow 1$	1	—	0	—	0	0	—	0	0	—	0	0
$0 \rightarrow 0$	0	0	—	0	—	0	0	—	0	0	—	0
Optional	—	—	—	—	—	—	—	—	—	—	—	—

The following example is used to review the reading of the maps for these five flip flops.

EXAMPLE:

		x_1			
		$x_1 x_2$			
		00	01	11	10
$y_3 y_4$	00	1	0	1	0
	01	1	1	0	1
	11	0	0	0	1
	10	1	0	1	0

FF_1

Figure 19-24

A pulse-input sequential circuit requires four flip flops. A map for the excitation of one of these flip flops is shown in Fig. 19-24. Obtain the expressions for pulsing the inputs of this flip flop, considering the five types: *S-R*, *S-R-SR*, *T*, *S-R-T*, and *S-R-SR-T*.

Flip flop

Type

Input

S-R

$$S_1 = x_1 \bar{y}_1 \bar{y}_2 \bar{y}_4 + x_1 \bar{y}_1 \bar{y}_3 y_4 = x_1 \bar{y}_1 (\bar{y}_2 \bar{y}_4 + \bar{y}_3 y_4)$$

$$R_1 = x_1 y_1 \bar{y}_2 \bar{y}_4 + x_1 y_1 y_2 y_4 = x_1 y_1 (\bar{y}_2 \bar{y}_4 + y_2 y_4)$$

S-R-SR

$$S_1 = x_1 \bar{y}_2 \bar{y}_4 + x_1 \bar{y}_3 y_4$$

$$R_1 = x_1 \bar{y}_2 \bar{y}_4 + x_1 y_2 y_4$$

T

$T_1 = x_1\bar{y}_2\bar{y}_4 + x_1\bar{y}_1\bar{y}_3y_4 + x_1y_1y_2y_4$
 $= x_1(\bar{y}_2\bar{y}_4 + \bar{y}_1\bar{y}_3y_4 + y_1y_2y_4)$

$S\text{-}R\text{-}T$

$S_1 = x_1\bar{y}_1\bar{y}_3y_4$
 $R_1 = x_1y_1y_2y_4$
 $T_1 = x_1\bar{y}_2\bar{y}_4$

$S\text{-}R\text{-}SR\text{-}T$

$S_1 = x_1\bar{y}_3y_4$
 $R_1 = x_1y_2y_4$
 $T_1 = x_1\bar{y}_2\bar{y}_4$

Sequential Circuit Outputs

The output expressions are independent of the type of flip flop used, and are read directly from the flow table with secondary assignment.

EXAMPLES:

In Fig. 19-25, the optional $y_1y_2 = 11$ combines with $y_1y_2 = 10$, to achieve some simplification. In Fig. 19-26, the optional $y_1y_2 = 10$ can not combine with $y_1y_2 = 01$, and no simplification results.

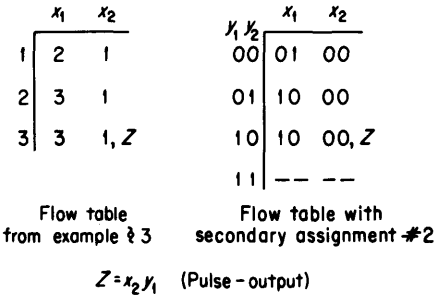


Figure 19-25

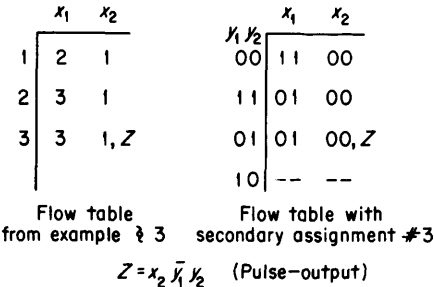


Figure 19-26

	x_1	x_2		y_1	y_2	x_1	x_2
1	2	1	00	11	00		
2	3	1	11	01	00		
3	4	1, Z	01	10	00, Z		
4	4	1	10	10	00		

Flow table from example § 4 Flow table with secondary assignment #3

$Z = x_2 \bar{y}_1 y_2$ (Pulse-output)

Figure 19-27

Figure 19-28 illustrates a poor choice of secondary assignment from the standpoint of economy of the output circuit. In Fig. 19-29, $y_1 y_2 = 01$ and $y_1 y_2 = 11$ combine to achieve simplification, resulting in a more economical output circuit.

	x_1	x_2	Z		y_1	y_2	x_1	x_2	Z
1	1	2	0	00	00	01	0		0
2	4	3	1	01	10	11	1		1
3	1	3	0	11	00	11	0		0
4	4	2	1	10	10	01	1		1

Flow table from example § 6 Flow table with secondary assignment #1

$Z = y_1 y_2 + y_1 \bar{y}_2$ (Level-output)

Figure 19-28

	x_1	x_2	Z		y_1	y_2	x_1	x_2	Z
1	1	2	0	00	00	01	0		0
2	4	3	1	01	11	10	1		1
3	1	3	0	10	00	10	0		0
4	4	2	1	11	11	01	1		1

Flow table from example § 6 Flow table with secondary assignment #2

$Z = y_2$ (Level-output)

Figure 19-29

Illustrative Circuit

Example §4, using secondary assignment #3 and *S-R-T* flip flops, is reviewed in its entirety, and the resulting circuit is shown (Figs. 19-30 to 19-32).

Circuit requirement: an output pulse *Z* is to be coincident with the first *x*₂ pulse immediately following exactly two consecutive *x*₁ pulses.

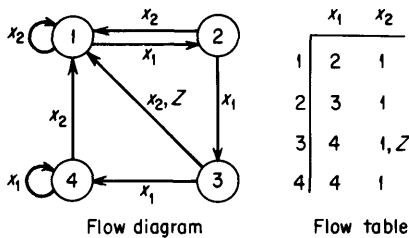
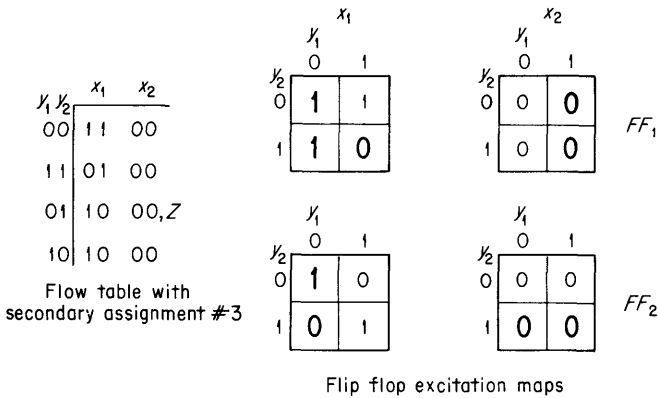


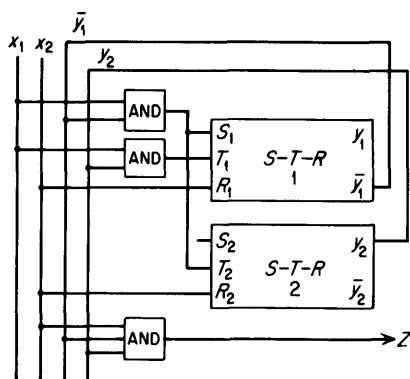
Figure 19-30



S-R-T flip flops:

$$S_1 = x_1 \bar{y}_1$$
$$R_1 = x_2$$
$$T_1 = x_1 y_2$$
$$S_2 = (\text{unused})$$
$$R_2 = x_2$$
$$T_2 = x_1 \bar{y}_1$$
$$Z = x_2 \bar{y}_1 y_2$$

Figure 19-31



Circuit diagram

Figure 19-32

Most-Economical Circuit Considerations

To obtain the most economical circuit, all types of secondary assignments and flip flops should generally be investigated. The same type of flip flop does not have to be used throughout; for example, FF_1 could be an *S-R* flip flop, and FF_2 a *T* flip flop.

To properly evaluate a given secondary assignment and flip flop combination, the entire circuit, that is, all flip flop excitation expressions and output expressions, must be examined as a whole, since there may be terms common to two or more of these expressions. Common terms should be looked for since they can lead to greater circuit economy. When there are alternate ways of reading a map, it may be advantageous to give preference to a common term, if it exists, even if it is larger than others.

The circuit cost is a function not only of the combinational circuitry referred to above, but also of the type of flip flop used, since the types of flip flops themselves can differ in cost.

PROBLEMS

1. From the following flow table with secondary assignment, obtain the flip flop excitation maps.

	x_1	x_2	Z
$y_1 y_2$			
00	10	00	0
10	01	11	1
01	01	00	0
11	10	11	1

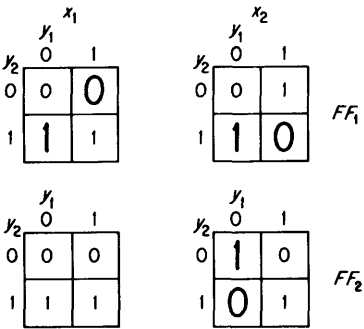
Figure 19-33

*2. From the following flow table with secondary assignment, obtain the flip flop excitation maps.

$y_1 y_2$	x_1	x_2
00	00	01
11	11	01, Z
01	11	10
10	00	10

Figure 19-34

3. Obtain the flip flop excitation expressions indicated below. Using the minimum combinational logic, draw the most economical circuit assuming that the flip flops are listed in order of increasing cost.



y_1
0 1
 y_2
0 1
0 1
1 1

0 0
1 1

 y_1
0 1
 y_2
0 1
0 1
1 01 0
0 1 FF_2

FF	Inputs	
S-R	$S_1 =$	$S_2 =$
	$R_1 =$	$R_2 =$
S-R-SR	$S_1 =$	$S_2 =$
	$R_1 =$	$R_2 =$
T	$T_1 =$	$T_2 =$
S-R-T	$S_1 =$	$S_2 =$
	$R_1 =$	$R_2 =$
	$T_1 =$	$T_2 =$
S-R-SR-T	$S_1 =$	$S_2 =$
	$R_1 =$	$R_2 =$
	$T_1 =$	$T_2 =$

$Z = y_2$

Figure 19-35

4. Design a sequential circuit for the requirements in Example § 2, Chapter 18.
 - (a) Using secondary assignment #1.
 - (b) Using secondary assignment #2.
 - (c) Using secondary assignment #3.

A solution requiring two two-input AND circuits as the total combinational circuit requirement is possible in all three cases.

5. Design a sequential circuit for the requirements in Example § 3, Chapter 18. A solution requiring two two-input AND circuits as the total combinational circuit requirement is possible.
- *6. Design a sequential circuit for the requirements in Example § 5, Chapter 18. A solution requiring one two-input AND circuits as the total combinational circuit requirement is possible.
- *7. Analyze the circuit in Fig. 19-36 and write a word statement describing the sequential circuit action. From the word statement, design a more economical circuit. A secondary assignment other than the one used in the circuit in Fig. 19-36 leads to a solution requiring only two two-input AND circuits as the total combinational circuit requirement.

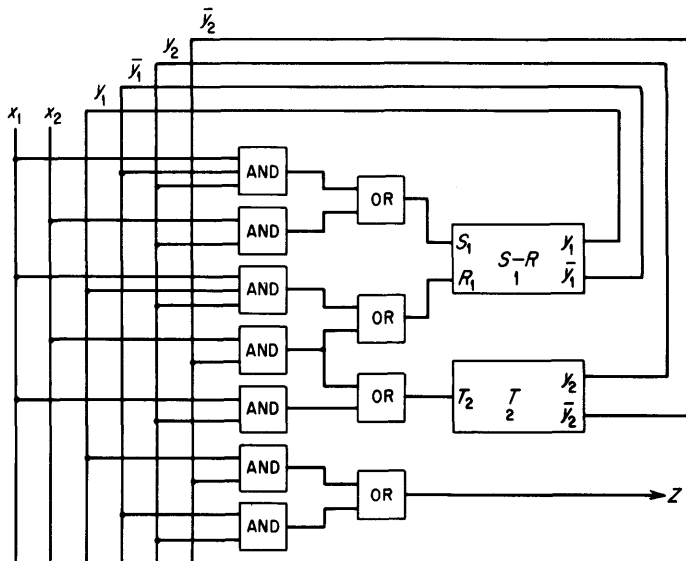


Figure 19-36

Related Literature for Further Study

Chapter 1

- G. Boole, *The Mathematical Analysis of Logic*, Cambridge, 1847.
- G. Boole, *An Investigation of the Laws of Thought*, London, 1854.
- W. H. Kautz, "A Survey and Assessment of Progress in Switching Theory and Logical Design in the Soviet Union," *IEEEEC*, Vol. EC-15, No. 2, pp. 164–204, April, 1966.
- C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE*, Vol. 57, pp. 713–723, 1938.

Chapter 2

- W. H. Burkhardt, "Theorem Minimization," *Proceedings of the Assoc. for Computing Machinery*, pp. 259–263, May 2–3, 1952.

Chapter 3

- "American Standard Graphic Symbols for Logic Diagrams," *American Standards Association*, ASA Y32.14-1962, Sept. 26, 1962. (Published by the AIEE)

- "Military Standard Graphic Symbols for Logic Diagrams," *MIL-STD-806B*, Feb. 26, 1962.
- T. J. Beatson, "Minimization of Components in Electronic Switching Circuits," *AIEE, Part I, Communication and Electronics*, Vol. 77, pp. 283-291, July, 1958.
- E. C. Nelson, "An Algebraic Theory for Use in Digital Computer Design," *IRETEC*, Vol. EC-3, No. 3, pp. 12-21, Sept., 1954.
- S. H. Washburn, "An Application of Boolean Algebra to the Design of Electronic Switching Circuits," *AIEE, Part I, Communication and Electronics*, Vol. 72, pp. 380-388, Sept., 1953.
- B. J. Yokelson and W. Ulrich, "Engineering Multi-Stage Diode Logic Circuits," *AIEE, Communication and Electronics*, No. 20, pp. 466-475, Sept., 1955.

Chapter 5

- P. Calingaert, "Multiple-Output Relay Switching Circuits," *Proceedings of an International Symposium on the Theory of Switching, Part II*, Harvard University, Cambridge, Mass., pp. 59-73, April, 1957.
- F. E. Hohn and L. R. Schissler, "Boolean Matrices and the Design of Combinational Relay Switching Circuits," *BSTJ*, Vol. 34, No. 1, pp. 177-202, Jan., 1955.
- F. E. Hohn, "A Matrix Method for the Design of Relay Circuits," *IRETCT* Vol. CT-2, No. 2, pp. 154-161, June, 1955.
- F. E. Hohn, "2N-Terminal Contact Networks," *Proceedings of an International Symposium on the Theory of Switching, Part II*, Harvard University, Cambridge, Mass., pp. 51-58, April, 1957.
- W. Keister, "The Logic of Relay Circuits," *AIEE Transactions*, Vol. 68, pp. 571-576, 1949.
- E. L. Lawler and G. A. Salton, "The Use of Parenthesis-Free Notation for the Automatic Design of Switching Circuits," *IRETEC*, Vol. EC-9, No. 3, pp. 342-352, Sept., 1960.
- R. E. Miller, "Formal Analysis and Synthesis of Bilateral Switching Networks," *IRETEC*, Vol. EC-7, No. 3, pp. 231-244, Sept., 1958.
- G. A. Montgomerie, "Sketch for an Algebra of Relay and Contactor Circuits," *J. IEE*, Vol. 95, No. 36, pp. 303-312, July, 1948.
- G. N. Povarov, "A Mathematical Theory for the Synthesis of Contact Networks with One Input and k Outputs," *Proceedings of an International Symposium on the Theory of Switching, Part II*, Harvard University, Cambridge, Mass., pp. 74-94, April, 1957.
- J. Riordan and C. E. Shannon, "The Number of Two-Terminal Series-Parallel Networks," *Journal of Mathematics and Physics*, Vol. 21, No. 2, pp. 83-93, 1942.
- V. N. Roginskij, "A Graphical Method for the Synthesis of Multiterminal Contact Networks," *Proceedings of an International Symposium on the Theory of Switch-*

- ing, *Part II*, Harvard University, Cambridge, Mass., pp. 302-315, April, 1957.
- B. D. Rudin, "A Theorem on SPDT Switching Circuits," *Proc. of the Western Joint Computer Conference*, pp. 129-132, March 1-3, 1955. (Published by the IRE.)
- A. H. Scheinman, "A Numerical-Graphical Method for Synthesizing Switching Circuits," *AIEE Transactions, Part I, Communication and Electronics*, pp. 687-689, 1957.
- A. H. Scheinman, "The Numerical-Graphical Method in the Design of Multi-terminal Switching Circuits," *AIEE Transactions, Part I, Communication and Electronics*, Vol. 78, pp. 515-519, Nov., 1959.
- W. Semon, "Matrix Methods in the Theory of Switching," *Proceedings of an International Symposium of the Theory of Switching, Part II*, Harvard University, Cambridge, Mass., pp. 13-50, April, 1957.
- W. Semon, "Synthesis of Series-Parallel Network Switching Functions," *BSTJ*, Vol. 37, No. 4, pp. 877-898, July, 1958.
- C. E. Shannon, "The Synthesis of Two-Terminal Switching Circuits," *BSTJ*, Vol. 28, No. 1, pp. 59-98, Jan., 1949.
- C. E. Shannon and E. F. Moore, "Machine Aid for Switching Circuit Design," *Proc. IRE*, Vol. 41, No. 10, pp. 1348-1351, Oct., 1953.
- R. A. Short, "The Design of Complementary-Output Networks," *IRETEC*, Vol. EC-11, No. 6, pp. 743-753, Dec., 1962.
- R. A. Short, "Correction to 'The Design of Complementary-Output Networks,'" *IEEEEC*, Vol. EC-12, No. 3, p. 232, June, 1963.

Chapter 6

- S. B. Akers, Jr., "A Truth Table Method for the Synthesis of Combinational Logic," *IRETEC*, Vol. EC-10, No. 4, pp. 604-615, Dec., 1961.
- T. C. Bartee, "The Automatic Design of Logical Networks," *Proc. of the Western Joint Computer Conference*, pp. 103-107, March 3-5, 1959. (Published by the IRE.)
- T. C. Bartee, "Computer Design of Multiple-Output Logical Networks," *IRETEC*, Vol. EC-10, No. 1, pp. 21-30, March, 1961.
- D.M.Y. Chang and T. H. Mott, Jr. "Computing Irredundant Normal Forms from Abbreviated Presence Functions," *IEEEEC*, Vol. EC-14, No. 3, pp. 335-342, June, 1965.
- A. K. Choudhury and M. S. Basu, "A Mechanized Chart for Simplification of Switching Functions," *IRETEC*, Vol. EC-11, No. 5, pp. 713-714, Oct., 1962.
- J. T. Chu, "A Generalization of a Theorem of Quine for Simplifying Truth Functions," *IRETEC*, Vol. EC-10, No. 2, pp. 165-168, June, 1961.
- S. R. Das and A. K. Choudhury, "Maxterm Type Expressions of Switching

- Functions and Their Prime Implications, "IEEETEC, Vol. EC-14, No. 6, pp. 920-923, Dec., 1965.
- B. Dunham and R. Fridshal, "The Problem of Simplifying Logical Expressions," *Journal of Symbolic Logic*, Vol. 24, No. 1, pp. 17-19, March, 1959.
- R. S. Gaines, "Implication Techniques for Boolean Functions," *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, S-164, pp. 174-182, Oct., 1964. (Published by the IEEE.)
- M. J. Ghazala (also Gazalé), "Irredundant Disjunctive and Conjunctive Forms of a Boolean Function," *IBM Journal of Research and Development*, Vol. 1, No. 2, pp. 171-176, April, 1957.
- J. F. Gimpel, "A Reduction Technique for Prime Implicant Tables," *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, S-164, pp. 183-191, Oct., 1964. (Published by the IEEE.)
- J. F. Gimpel, "A Method of Producing a Boolean Function Having an Arbitrarily Prescribed Prime Implicant Table," *IEEETEC*, Vol. EC-14, No. 3, pp. 485-488, June, 1965.
- J. F. Gimpel, "A Reduction Technique for Prime Implicant Tables," *IEEETEC*, Vol. EC-14, No. 4, pp. 535-541, Aug., 1965.
- F. B. Hall, "Boolean Prime Implicants by the Binary Sieve Method," *AIEE Transactions, Part I, Communication and Electronics*, Vol. 80, pp. 709-713, Jan., 1962.
- B. Harris, "An Algorithm for Determining Minimal Representations of a Logic Function," *IRETEC*, Vol. EC-6, No. 2, pp. 103-108, June, 1957.
- R. Hockney, "An Intersection Algorithm Giving All Irredundant Forms from a Prime Implicant List," *IEEETEC*, Vol. EC-11, No. 2, pp. 289-290, April, 1962.
- Z. Kohavi, "Minimizing of Incompletely Specified Sequential Switching Circuits," *Office of Technical Services Government Research Report AD 286,174*, May 10, 1962.
- F. Luccio, "A Method for the Selection of Prime Implicants," *IEEETEC*, Vol. EC-15, No. 2, pp. 205-212, April, 1966.
- E. J. McCluskey, Jr., "Minimization of Boolean Functions," *BSTJ*, Vol. 35, No. 6, pp. 1417-1444, Nov., 1956.
- E. J. McCluskey, Jr. "Minimal Sums for Boolean Functions Having Many Unspecified Fundamental Products," *Proceedings of the Second Annual Symposium on Switching Circuit Theory and Logical Design*, pp. 10-17, Sept., 1961. (Published by the AIEE.)
- R. McNaughton and B. Mitchell, "The Minimality of Rectifier Nets with Multiple Outputs Incompletely Specified," *Journal of the Franklin Institute*, Vol. 264, No. 6, pp. 457-480, Dec., 1957.
- A. R. Meo, "On the Determination of the ps Maximal Implicants of a Switching Function," *IEEETEC*, Vol. EC-14, No. 6, pp. 830-840, Dec., 1965.
- F. Mileto and G. Putzolu, "Average Values of Quantities Appearing in Boolean

- Function Minimization," *IEEEEC*, Vol. EC-13, No. 2, pp. 87-92, April, 1964.
- F. Mileto and G. Putzolu, "Average Values of Quantities Appearing in Multiple Output Boolean Minimization," *IEEEEC*, Vol. EC-14, No. 4, pp. 542-552, Aug., 1965.
- H. Mott and C. C. Carroll, "Numerical Procedures for Boolean Function Minimization," *IEEEEC*, Vol. EC-13, No. 4, p. 470, Aug., 1964.
- T. H. Mott, Jr., "Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants," *IRETEC*, Vol. EC-9, No. 2, pp. 245-252, June, 1960.
- R. Mueller, "On the Synthesis of a Minimal Representation of a Logic Function," *Air Force Cambridge Research Center Technical Report 55-104*, April 1955.
- D. E. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection," *IRETEC*, Vol. EC-3, No. 3, pp. 6-12, Sept., 1954.
- D. E. Muller, "Complexity in Electronic Switching Circuits," *IRETEC*, Vol. EC-5, No. 1, pp. 15-19, March, 1956.
- R. J. Nelson, "Weak Simplest Normal Truth Functions," *Journal of Symbolic Logic*, Vol. 20, No. 3, pp. 232-234, Sept., 1955.
- R. J. Nelson, "Simplest Normal Truth Functions," *Journal of Symbolic Logic*, Vol. 20, No. 2, pp. 105-108, June, 1955.
- A. J. Nichols and A. J. Bernstein, "State Assignments in Combinational Networks," *IEEEEC*, Vol. EC-14, No. 3, pp. 343-349, June, 1965.
- S. R. Petrick, "A Direct Determination of the Irredundant Forms of a Boolean Function from the Set of Prime Implicants," *Air Force Cambridge Research Center Technical Report 56-110*, April, 1956.
- R. B. Polansky, "Further Notes on Simplifying Multiple-output Switching Circuits," *Electronics Systems Laboratory Mem. 7849-M-330*, M.I.T., Cambridge, Mass., pp. 1-6, Oct. 26, 1959.
- R. B. Polansky, "Minimization of Multiple-Output Switching Circuits," *AIEE Transactions, Part I, Communication and Electronics*, Vol. 80, pp. 67-73, March, 1961.
- I. B. Pyne and E. J. McCluskey, Jr. "The Reduction of Redundancy in Solving Prime Implicant Tables," *IRETEC*, Vol. EC-11, No. 4, pp. 473-482, Aug., 1962.
- W. V. Quine, "The Problem of Simplifying Truth Functions," *American Mathematical Monthly*, Vol. 59, pp. 521-531, Oct., 1952.
- W. V. Quine, "A Way to Simplify Truth Functions," *American Mathematical Monthly*, Vol. 62, pp. 627-631, Nov., 1955.
- W. V. Quine, "On Cores and Prime Implicants of Truth Functions," *American Mathematical Monthly*, Vol. 66, pp. 755-760, Nov., 1959.
- T. Rado, "Comments on the Presence Function of Gazalé," *IBM Journal of Research and Development*, Vol. 6, No. 2, pp. 268-269, April, 1962.
- J. P. Roth, "Algebraic Topological Methods in Synthesis," *Proceedings of an Inter-*

- national Symposium on the Theory of Switching, Part I*, Harvard University, Cambridge, Mass., pp. 57-73, April, 1959.
- J. P. Roth and E. G. Wagner, "Algebraic Topological Methods for the Synthesis of Switching Systems, Part III: Minimization of Nonsingular Boolean Trees," *IBM Journal of Research and Development*, Vol. 3, No. 4, pp. 326-344, Oct., 1959.
- J. P. Roth, "Minimization Over Boolean Trees," *IBM Journal of Research and Development*, Vol. 4, No. 5, pp. 543-558, Nov., 1960.
- J. P. Roth and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal of Research and Development*, Vol. 6, No. 2, pp. 227-238, April, 1962.
- E. W. Samson and B. E. Mills, "Circuit Minimization: Algebra and Algorithm for new Boolean Canonical Expressions," *Air Force Cambridge Research Center Technical Report 54-21*, April, 1954.
- E. W. Samson and R. Mueller, "Circuit Minimization: Minimal and Irredundant Boolean Sums by Alternative Set Method," *Air Force Cambridge Research Center Technical Report 55-109*, June, 1955.
- A. H. Scheinman, "A Method for Simplifying Boolean Functions," *BSTJ*, Vol. 41, No. 4, pp. 1337-1346, July, 1962.
- T. Singer, "Some Uses of Truth Tables," *Proceedings of an International Symposium on the Theory of Switching, Part I*, Harvard University, Cambridge, Mass. pp. 125-133, April, 1957.
- R. H. Urbano and R. K. Mueller, "A Topological Method for the Determination of the Minimal Forms of a Boolean Function," *IRETEC*, Vol. EC-5, No. 3, pp. 126-132, Sept., 1956.
- G. C. Vandling, "The Simplification of Multiple-Output Networks Composed of Unilateral Devices," *IRETEC*, Vol. EC-9, No. 4, pp. 477-486, Dec., 1960.
- J. N. Warfield, "A Note on the Reduction of Switching Functions," *IRETEC*, Vol. EC-7, No. 2, pp. 180-181, June, 1958.

Chapter 7

- M. E. Arthur, "Geometric Mapping of Switching Functions," *IRETEC*, Vol. EC-10, No. 4, pp. 631-637, Dec., 1961.
- T. M. Booth, "The Vertex-Frame Method for Obtaining Minimal Proposition-Letter Formulas," *IRETEC*, Vol. EC-11, No. 2, pp. 144-154, April, 1962.
- M. Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits," *Trans. AIEE, Part I, Communication and Electronics*, Vol. 72, pp. 593-599, Nov., 1953.
- E. W. Veitch, "A Chart Method for Simplifying Truth Functions," *Proceedings of the Assoc. for Computing Machinery*, pp. 127-133, May 2-3, 1952.

Chapter 8

- D. R. Brown and N. Rochester, "Rectifier Networks for Multiposition Switching," *Proc. IRE*, Vol. 37, No. 2, pp. 139-147, Feb., 1949.

- A. W. Burks, *et al.*, "The Folded Tree," *Journal of the Franklin Institute*, Vol. 260, Part I, No. 1, pp. 9–24, July, 1955; Part II, No. 2, pp. 115–126, Aug, 1955.
- E. L. Lawler, "The Minimal Synthesis of Tree Structures," *Proc. of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, S-156, pp. 63–82, Sept., 1963. (Published by the IEEE.)
- M. P. Marcus, "Minimization of the Partially-Developed Transfer Tree," *IRETEC*, Vol. EC-6, No. 2, pp. 92–95, June, 1957.
- E. F. Moore, "Minimal Complete Relay Decoding Networks," *IBM Journal of Research and Development*, Vol. 4, No. 5, pp. 525–531, Nov., 1960.
- S. H. Washburn, "Relay 'Trees' and Symmetric Circuits," *Trans. AIEE*, Part I, Vol. 68, pp. 582–586, 1949.

Chapter 9

- R. F. Arnold and M. A. Harrison, "Algebraic Properties of Symmetric and Partially Symmetric Boolean Functions," *IEEEEC*, Vol. EC-12, No. 3, pp. 244–251, June, 1963.
- S. H. Caldwell, "The Recognition and Identification of Symmetric Switching Functions," *Trans. AIEE*, Part II, Vol. 73, pp. 142–147, May, 1954.
- B. Elspas, "Self-Complementary Symmetry Types of Boolean Functions," *IRETEC*, Vol. EC-9, No. 2, pp. 264–266, June, 1960.
- G. Epstein, "Synthesis of Electronic Circuits for Symmetric Functions," *IRETEC*, Vol. EC-7, No. 1, pp. 57–60, March, 1958.
- M. P. Marcus, "The Detection and Identification of Symmetric Switching Functions with the Use of Tables of Combinations," *IRETEC*, Vol. EC-5, No. 4, pp. 237–239, Dec., 1956.
- E. J. McCluskey, Jr., "Detection of Group Invariance or Total Symmetry of a Boolean Function" *BSTJ*, Vol. 35, No. 6, pp. 1445–1453, Nov., 1956.
- A. Mukhopadhyay, "Detection of Total or Partial Symmetry of a Switching Function with the Use of Decomposition Charts," *IEEEEC*, Vol. EC-12, No. 5, pp. 553–557, Oct., 1963.
- C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE*, Vol. 57, pp. 713–723, 1938.
- C. L. Sheng, "Detection of Totally Symmetric Boolean Functions," *IEEEEC*, Vol. EC-14, No. 6, pp. 924–926, Dec., 1965.
- S. H. Washburn, "Relay 'Trees' and Symmetric Circuits," *Trans. AIEE*, Part I, Vol. 68, pp. 582–586, 1949.

Chapter 10

- D. L. Epley, "Design of Combinational Switching Circuits Using an Iterative Configuration," *Office of Technical Services Government Research Report AD 289 309*.
- F. C. Hennie, "Analysis of Bilateral Iterative Networks," *IRETCT*, Vol. CT-6, No. 1, pp. 35–45, March, 1959.

F. C. Hennie, *Iterative Arrays of Logical Circuits*, The M.I.T. Press and John Wiley & Sons, Inc., New York, 1961.

E. J. McCluskey, Jr., "Iterative Combinational Switching Networks—General Design Considerations," *IRETEC*, Vol. EC-7, No. 4, pp. 285–291, Dec., 1958.

Chapter 12

R. W. Hamming, "Error Detecting and Error Correcting Codes," *BSTJ*, Vol. 29, No. 2, pp. 147–160, April, 1950.

W. W. Peterson, *Error-Correcting Codes*, The M.I.T. Press and John Wiley & Sons, Inc., New York, 1961.

Chapter 13

K. E. Batcher, "On the Number of Stable States in a NOR Network," *IEEEEC*, Vol. EC-14, No. 6, pp. 931–932, Dec., 1965.

W. S. Bennett, "Minimizing and Mapping Sequential Circuits," *AIEE Communication and Electronics*, pp. 443–447, Sept., 1955.

J. A. Brzozowski, "A Survey of Regular Expressions and Their Applications," *IRETEC*, Vol. EC-11, No. 3, pp. 324–335, June, 1962.

J. A. Brzozowski, "Some Problems in Relay Circuit Design," *IEEEEC*, Vol. EC-14, No. 4, pp. 630–634, Aug., 1965.

A. W. Burks and J. B. Wright, "Theory of Logical Nets," *Proc. IRE*, Vol. 41, No. 10, pp. 1357–1365, Oct., 1953.

A. W. Burks and H. Wang, "The Logic of Automata, Parts I and II," *JACM*, Vol. 4, No. 2, pp. 193–218, April, 1957; No. 3, pp. 279–297, July, 1957.

B. Elspas, "The Theory of Autonomous Linear Sequential Networks," *IRETCT*, Vol. CT-6, No. 1, pp. 45–60, March, 1959.

B. Friedland, "Linear Modular Sequential Circuits," *IRETCT*, Vol. CT-6, No. 1, pp. 61–68, March, 1959.

J. Hartmanis, "Linear Multivalued Sequential Coding Networks," *IRETCT*, Vol. CT-6, No. 1, pp. 69–74, March, 1959.

D. A. Huffman, "The Synthesis of Sequential Circuits," *Journal of the Franklin Institute*, Vol. 257, No. 3, pp. 161–190, March, 1954; No. 4 pp. 275–303, April, 1954.

D. A. Huffman, "A Study of the Memory Requirements of Sequential Switching Circuits," *Research Lab. of Electronics Technical Report 293*, M.I.T., March 14, 1955.

M. Kliman and O. Lowenschuss, "Asynchronous Electronic Switching Circuits," *IRE National Conventional Record*, Part 4, pp. 267–274, 1959.

G. H. Mealy, "A Method for Synthesizing Sequential Circuits," *BSTJ*, Vol. 34, No. 5, pp. 1045–1079, Sept., 1955.

- E. F. Moore, "Gedanken-Experiments on Sequential Machines," *Automata Studies*, pp. 129–153, Princeton University Press, Princeton, N.J., 1956.
- D. E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits," *Proceedings of an International Symposium on the Theory of Switching, Part I*, Harvard University, Cambridge, Mass., pp. 204–243, April, 1957.
- G. Ott and N. H. Feinstein, "Design of Sequential Machines from their Regular Expressions," *JACM*, Vol. 8, pp. 585–600, Oct., 1961.
- A. E. Ritchie, "Sequential Aspects of Relay Circuits," *AIEE Transactions*, Part I, Vol. 68, pp. 577–581, 1949.
- J. M. Simon, "Some Aspects of the Network Analysis of Sequence Transducers," *Journal of the Franklin Institute*, Vol. 265, No. 6, pp. 439–450, June, 1958.
- J. M. Simon, "A Note on Memory Aspects of Sequence Transducers," *IRETCT*, Vol. CT-6, No. 1, pp. 26–29, March, 1959.
- F. S. Stănculescu, "Sequential Logic and Its Application to the Synthesis of Finite Automata," *IEEEEC*, Vol. EC-14, No. 6, pp. 786–791, Dec., 1965.
- S. H. Unger, "A Study of Asynchronous Logical Feedback Networks," *Research Lab. of Electronics Technical Report 320*, M.I.T., April 26, 1957.
- N. Zierler, "Several Binary-Sequence Generators," *Lincoln Lab. Technical Report 95*, M.I.T., Sept. 12, 1955.

Chapters 14 and 15

- D. D. Aufenkamp and F. E. Hohn, "Analysis of Sequential Machines," *IRETEC*, Vol. EC-6, No. 4, pp. 276–285, Dec., 1957.
- D. D. Aufenkamp, "Analysis of Sequential Machines II," *IRETEC*, Vol. EC-7, No. 4, pp. 299–306, Dec., 1958.
- H. Frank and S. S. Yau, "Improving Reliability of a Sequential Machine by Error-Correcting State Assignments," *IEEEEC*, Vol. EC-15, No. 1, pp. 111–113, Feb., 1966.
- A. Gill, "A Note on Moore's Distinguishability Theorem," *IRETEC*, Vol. EC-10, No. 2, pp. 290–291, June, 1961.
- S. Ginsburg, "A Synthesis Technique for Minimal State Sequential Machines," *IRETEC*, Vol. EC-8, No. 1, pp. 13–24, March, 1959.
- S. Ginsburg, "A Technique for the Reduction of a Given Machine to a Minimal-State Machine," *IRETEC*, Vol. EC-8, No. 3, pp. 346–355, Sept., 1959.
- S. Ginsburg, "Synthesis of Minimal-State Machines," *IRETEC*, Vol. EC-8, No. 4, pp. 441–449, Dec., 1959.
- A. Grasselli, "Minimal Closed Partitions for Incompletely Specified Flow Tables," *IEEEEC*, Vol. EC-15, No. 2, pp. 245–249, April, 1966.
- A. Grasselli and F. Luccio, "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks," *IEEEEC*, Vol. EC-14, No. 3, pp. 350–359, June, 1965.

- J. Hartmanis, "Symbolic Analysis of a Decomposition of Information Processing Machines," *Information and Control*, Vol. 3, No. 2, pp. 154-178, June, 1960.
- J. Hartmanis, "Further Results on the Structure of Sequential Machines," *JACM*, Vol. 10, No. 1, pp. 78-88, Jan., 1963.
- M. P. Marcus, "Derivation of Maximal Compatibles Using Boolean Algebra," *IBM Journal of Research and Development*, Vol. 8, No. 5, pp. 537-538, Nov., 1964.
- E. J. McCluskey, Jr., "Minimum-State Sequential Circuits for a Restricted Class of Incompletely Specified Flow Tables," *BSTJ*, Vol. 41, No. 6, pp. 1759-1768, Nov., 1962.
- R. Narasimhan, "Minimizing Incompletely Specified Sequential Switching Functions," *IRETEC*, Vol. EC-10, No. 3, pp. 531-532, Sept., 1961.
- D. B. Netherwood, "Minimal Sequential Machines," *IRETEC*, Vol. EC-8, No. 3, pp. 339-345, Sept., 1959.
- M. C. Paull and S. H. Unger, "Minimizing the Number of States in Sequential Switching Functions," *IRETEC*, Vol. EC-8, No. 3, pp. 356-367, Sept., 1959.
- M. O. Rabin and D. Scott, "Finite Automata and Their Decision Problems," *IBM Journal of Research and Development*, Vol. 3, No. 2, pp. 114-125, April, 1959.
- I. S. Reed, "Some Remarks on State Reduction of Asynchronous Circuits by the Paull-Unger Method," *IEEEEC*, Vol. EC-14, No. 2, pp. 262-265, April, 1965.
- S. H. Unger, "Flow Table Simplification—Some Useful Aids," *IEEEEC*, Vol. EC-14, No. 3, pp. 472-475, June, 1965.

Chapter 16

(See Chapter 18 for many related references.)

Chapter 17

- E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM Journal of Research and Development*, Vol. 9, No. 2, pp. 90-99, March, 1965.
- D. A. Huffman, "The Design and Use of Hazard-Free Switching Networks," *JACM*, Vol. 4, No. 1, pp. 47-62, Jan., 1957.
- M. P. Marcus, "Relay Essential Hazards," *IEEEEC*, Vol. EC-12, No. 4, pp. 405-407, Aug., 1963.
- D. E. Muller, "Treatment of Transition Signals in Electronic Switching Circuits by Algebraic Methods," *IRETEC*, Vol. EC-8, No. 3, p. 401, Sept., 1959.
- S. H. Unger, "Hazards and Delays in Asynchronous Sequential Switching Circuits," *IRETCT*, Vol. CT-6, No. 1, pp. 12-25, March, 1959.
- M. Yoeli and S. Rinon, "Application of Ternary Algebra to the Study of Static Hazards," *JACM*, Vol. 11, No. 1, pp. 84-97, Jan., 1964.

Chapter 18

- D. B. Armstrong, "A Programmed Algorithm for Assigning Internal Codes to Sequential Machines," *IRETEC*, Vol. EC-11, No. 4, pp. 466-472, Aug., 1962.
- D. B. Armstrong, "On the Efficient Assignment of Internal Codes to Sequential Machines," *IRETEC*, Vol. EC-11, No. 5, pp. 611-622, Oct., 1962.
- R. Bianchini and C. Freiman, "On Internal Variable Assignments for Sequential Switching Circuits," *IRETEC*, Vol. EC-10, No. 1, pp. 95-96, March, 1961.
- R. C. Brigham, "Some Properties of Binary Counters with Feedback," *IRETEC*, Vol. EC-10, No. 4, pp. 699-701, Dec., 1961.
- F. M. Brown, "Code Transformation in Sequential Machines," *IEEEETEC*, Vol. EC-14, No. 6, pp. 822-829, Dec., 1965.
- J. A. Brzozowski and E. J. McCluskey, Jr., "Signal Flow Graph Techniques for Sequential Circuit State Diagrams," *IEEEETEC*, Vol. EC-12, No. 2, pp. 67-76, April, 1963.
- W. J. Cadden, "Equivalent Sequential Circuits," *IRETCT*, Vol. CT-6, No. 1, pp. 30-34, March, 1959.
- W. H. Davidow, "A State Assignment Technique for Synchronous Sequential Networks," *Stanford Electronics Laboratories Technical Report* 1901-1, Stanford University, July 20, 1961.
- T. A. Dolotta and E. J. McCluskey, Jr., "The Coding of Internal States of Sequential Circuits," *IEEEETEC*, Vol. EC-13, No. 5, pp. 549-562, Oct, 1964.
- J. Hartmanis, "On the State Assignment Problem for Sequential Machines. I," *IRETEC*, Vol. EC-10, No. 2, pp. 157-165, June, 1961.
- J. Hartmanis, "The Equivalence of Sequential Machine Models," *IEEEETEC*, Vol. EC-12, No. 1, Feb., 1963.
- J. Hartmanis, "Two Tests for the Linearity of Sequential Machines," *IEEEETEC*, Vol. EC-14, No. 6, pp. 781-786, Dec., 1965.
- R. M. Karp, "Some Techniques of State Assignment for Synchronous Sequential Machines," *IEEEETEC*, Vol. EC-13, No. 5, pp. 507-518, Oct., 1964.
- R. M. Karp, "Correction to 'Some Techniques of State Assignment for Synchronous Sequential Machines,'" *IEEEETEC*, Vol. EC-14, No. 1, p. 61, Feb., 1965.
- Z. Kohavi, "Secondary State Assignment for Sequential Machines," *IEEEETEC*, Vol. EC-13, No. 3, pp. 193-203, June, 1964.
- Z. Kohavi, "Reduction of Output Dependency in Sequential Machines," *IEEEETEC*, Vol. EC-14, No. 6, pp. 932-934, Dec, 1965.
- M. P. Marcus, "Cascaded Binary Counters with Feedback," *IEEEETEC*, Vol. EC-12, No. 4, pp. 361-364, Aug., 1963.
- E. J. McCluskey, Jr. and S. H. Unger, "A Note on the Number of Internal Assignments for Sequential Switching Circuits," *IRETEC*, Vol. EC-8, No. 4, pp. 439-440, Dec., 1959.

- A. J. Nichols, "Comments on Armstrong's State Assignment Techniques," *IEEEEC*, Vol. EC-12, No. 4, pp. 407-409, Aug. 1963.
- S. Seshu, R. E. Miller, and G. Metze, "Transition Matrices of Sequential Machines," *IRETCT*, Vol. CT-6, No. 1, pp. 5-12, March, 1959.
- R. E. Stearns and J. Hartmanis, "On the State Assignment Problem for Sequential Machines II," *IRETEC*, Vol. EC-10, No. 4, pp. 593-603, Dec., 1961.
- T. U. Zahle, "On Coding the States of Sequential Machines with the Use of Partition Pairs," *IEEEEC*, Vol. EC-15, No. 2, pp. 249-253, April, 1966.

Answers and Solutions to Problems

Chapter 1

1. (a) 1
(b) 0
(c) 1
(d) \bar{C}
(e) \bar{C}
(f) 0
2. (a) $[(A + \bar{B}) C + \bar{D}] E + \bar{F}$
(b) $[\bar{S} + W(I + \bar{T}C)]\bar{H}$
3. (a) $(A + E) C (DF + B)$
(b) $BF + E + (A + C) D$
(c) $B(D + E)[AC + F(G + H)]$
(d) $CE + F + (A + B)(D + GH)$
(e) $A(C + D)[B(\bar{E} + F) + \bar{G}\bar{H}]$
(f) $\bar{C} + EF + (A\bar{B} + D)(\bar{G} + \bar{H}K)$

4. (a) $A\bar{C}$
 (b) $A\bar{C} + A\bar{B} + \bar{C}D$
 (c) $(\bar{A} + B)(B + CD) = B + \bar{A}CD$
 (d) $\bar{H}E + DE + G\bar{H} + HF\bar{E}$
 (e) $(L + \bar{P})(L + M)(Q + P + \bar{L})(\bar{P} + N)$
 (f) $(A + BC)(A + D)(BC + E)(\bar{A} + \bar{B} + \bar{C} + F)$
5. (a) $AB + \bar{B}C\bar{D} + C\bar{D}E$
 (b) $B\bar{C}\bar{E} + AE$
 (c) $(A + B)(C + \bar{D})$
 (d) $(B + \bar{C} + \bar{D})(A + D)$
 (e) $AD + B\bar{C}$
 (f) $\bar{A}B\bar{C} + CD$
 (g) $(\bar{A} + B + \bar{C})(C + D)(\bar{A} + B + E)$
 (h) $(\bar{A} + B + \bar{C})(C + D)$
6. (a) $AC + \bar{B}\bar{A} + \bar{D}\bar{B} + \bar{C}EB + \bar{G}C$
 (b) $(P + I)(I + \bar{T})(P + A)(\bar{P} + O + T)(U + \bar{T})$
 (c) $CE + \bar{D}\bar{E} + BC$
 (d) $(\bar{A} + B)(C + A)(B + D)$
 (e) $AF + \bar{E}\bar{F} + A\bar{B} + AD$
 (f) $\bar{K}L + \bar{L}M + HM + \bar{G}\bar{M}$
 (g) $X\bar{Y} + \bar{X}Z + \bar{Y}Z + X\bar{Z} = X\bar{Y} + \bar{X}Z + X\bar{Z}$ or $\bar{X}Z + \bar{Y}Z + X\bar{Z}$
 (h) $(\bar{A} + B)(A + \bar{C})(B + \bar{C})(C + \bar{A}) = (\bar{A} + B)(A + \bar{C})(C + \bar{A})$ or $(A + \bar{C})(B + \bar{C})(C + \bar{A})$
7. (a) $(A + \bar{D}E)(\bar{A} + B + \bar{C})$
 (b) $[\bar{D} + E(F + G)](D + A + \bar{B}\bar{C})$
8. (a) $A(\bar{D} + \bar{E}) + \bar{A}BC$
 (b) $\bar{D}(E + F)G + D(A\bar{B} + \bar{C})$
9. (a) $(A + BC + D + E)(\bar{A} + G + C + FE)$
 or $A(G + C + FE) + \bar{A}(BC + D + E)$
 (b) $[A + \bar{B}(\bar{D} + E)(G + \bar{H} + J)][\bar{A} + C(\bar{D} + E + F)(G + \bar{H})]$
 or $AC(\bar{D} + E + F)(G + \bar{H}) + \bar{A}\bar{B}(\bar{D} + E)(G + \bar{H} + J)$
10. $\bar{A}C + \bar{A}B + A\bar{C} + A\bar{B} + \bar{B}C + B\bar{C} =$
 $\bar{A}\bar{B} + \bar{B}C + A\bar{C}$
 $(\bar{A} + \bar{C})(A + B) + \bar{B}C$
 $(\bar{B} + \bar{A})(B + C) + A\bar{C}$
 $(\bar{C} + \bar{B})(C + A) + \bar{A}B$
 $\bar{A}(B + C) + A(\bar{B} + \bar{C})$
 $\bar{B}(A + C) + B(\bar{A} + \bar{C})$
 $\bar{C}(A + B) + C(\bar{A} + \bar{B})$
 $(\bar{A} + \bar{B} + \bar{C})(A + B + C)$
 $A\bar{B} + B\bar{C} + \bar{A}C$
 $(A + C)(\bar{A} + \bar{B}) + B\bar{C}$
 $(B + A)(\bar{B} + \bar{C}) + \bar{A}C$
 $(C + B)(\bar{C} + \bar{A}) + \bar{A}B$

Chapter 2

1. (a) $(\bar{A} + B)(\bar{A} + \bar{C})$
 (b) $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$
 (c) $(\bar{A} + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C)$
2. (a) $AB + A\bar{C} + D$
 (b) $(A + D)(B + \bar{C} + D)$
 (c) $AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD$
 (d) $(A + \bar{B} + \bar{C} + D)(A + \bar{B} + C + D)(A + B + \bar{C} + D)(A + B + C + D)(\bar{A} + B + \bar{C} + D)$

Chapter 5

1.

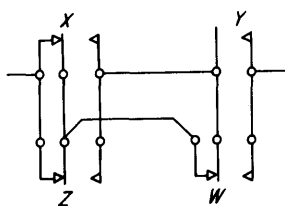


Figure 5-1A

3.

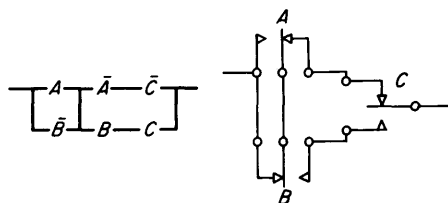


Figure 5-3A

4.

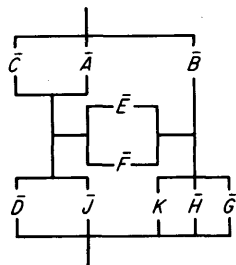


Figure 5-4A

5.

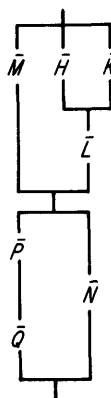


Figure 5-5A

8.

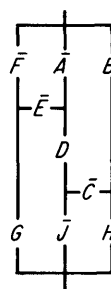


Figure 5-8A

9.

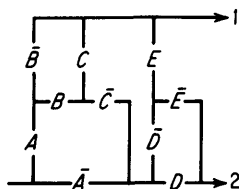


Figure 5-9A

10.

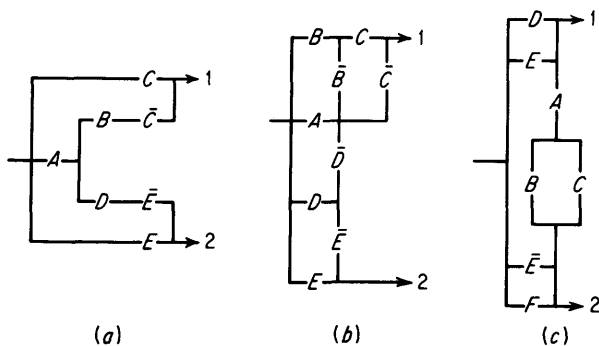


Figure 5-10A

11.

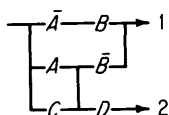


Figure 5-11A

Chapter 6

1. $\bar{A}\bar{B} + A\bar{C}D + AB\bar{D}$
2. $(A + \bar{B})(\bar{A} + \bar{C} + \bar{D})(B + C + D)$ or $(A + \bar{B})(\bar{A} + \bar{C} + \bar{D})(\bar{A} + B + D)$
3. $\bar{A}\bar{C}\bar{D}, \bar{A}B\bar{C}, \bar{A}CD, \bar{A}BD, B\bar{C}D, AC\bar{D}, \bar{B}\bar{D}, \bar{B}C$
4. (a) $7(UV + UWX + UXY + VWZ + VYZ + WXZ + XYZ)$
 (b) $\bar{A}\bar{B}\bar{E} + \bar{A}\bar{C}E$
 (c) $BE + D + C\bar{E}$
6. 1: $\bar{A}B\bar{C}\bar{D} + \bar{B}\bar{D} + BCD$
 2: $\bar{A}BCD + \bar{A}\bar{C}D + \bar{B}\bar{D}$
 3: $\bar{A}\bar{B}C + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD$

Chapter 7

1. $A + \bar{D} + \bar{B}C$
3. $\bar{A}B + \bar{C}D + B\bar{C} + \bar{A}D$ or $(\bar{A} + \bar{C})(B + D)$
4. $B\bar{D} + \bar{C}D + \bar{A}D$ or $(B + D)(\bar{A} + \bar{C} + \bar{D})$
6. $\bar{A}B\bar{C} + ABD\bar{E} + \bar{B}C\bar{D} + AB\bar{D} + \bar{A}\bar{C}\bar{E}$
 or $\bar{A}B\bar{C} + ABD\bar{E} + \bar{B}C\bar{D} + AB\bar{D} + \bar{B}\bar{D}\bar{E}$
 or $\bar{A}B\bar{C} + ABD\bar{E} + \bar{B}C\bar{D} + AB\bar{D} + \bar{B}\bar{C}\bar{E}$
7. $(\bar{B} + D)(B + \bar{D})(A + C + \bar{E} + D)(\bar{A} + \bar{E} + \bar{B})(A + \bar{C} + \bar{D})$
 or $(\bar{B} + D)(B + \bar{D})(A + C + \bar{E} + B)(\bar{A} + \bar{E} + \bar{D})(A + \bar{C} + \bar{B})$
8. $\bar{C}\bar{D}\bar{E}\bar{F} + C\bar{D}\bar{E}\bar{F} + \bar{A}CD\bar{F} + \bar{A}\bar{C}D\bar{E} + AB\bar{D} + \bar{B}\bar{D}EF$

Chapter 8

1. Many solutions with 1-7-7-8-8 distribution.
- 2.

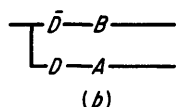
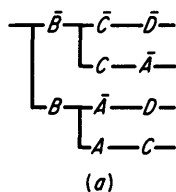


Figure 8-2A

3. $(2^n - 1) - (m - p)$

(a) $15 - (8 - 4) = 11$

(b) $15 - (8 - 4) = 11$

(c) $15 - (8 - 3) = 10$

(d) $15 - (8 - 3) = 10$

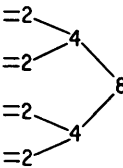
5.



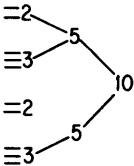
$160 - 96 = 64$



$384 - 176 = 208$



$2048 - 608 = 1440$



$10,240 - 2240 = 8000$

Figure 8-5A

Chapter 9

1. (a)

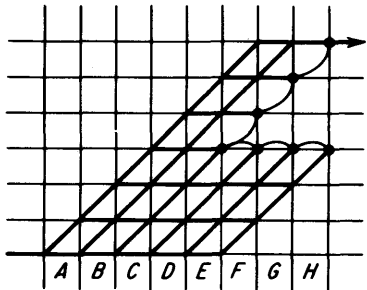


Figure 9-1A

(b)

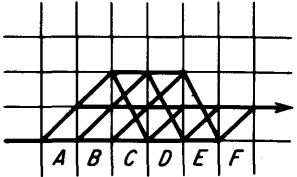


Figure 9-1A

(c) $S_{1,4}^7 ABCDEFG = S_{3,6}^7 \bar{A} \bar{B} \bar{C} \bar{D} \bar{E} \bar{F} \bar{G}$

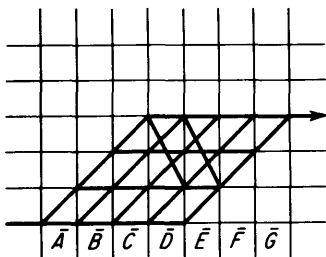


Figure 9-1A

(d)

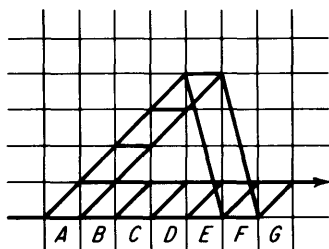


Figure 9-1A

2. (a) $S_{1,2,3,4}^4 ABCD$
 (b) $S_{0,1,2,3,4}^4 EFGH = 1$
 (c) 0
 (d) $S_{0,1,3}^4 MNOP$
 (e) $S_{1,2}^4 \bar{Q} \bar{R} \bar{S} \bar{T}$
 (f) $S_{0,2,4}^4 \bar{U} \bar{V} \bar{W} \bar{X}$

3.

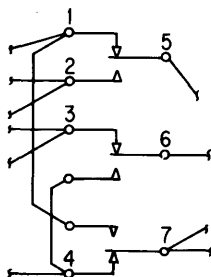


Figure 9-3A

5. (a) $S_{3,5}^5 A \bar{B} C D \bar{E}$
 (b) $S_{2,4}^5 A \bar{B} C D$
 (c) $S_{1,3}^5 A \bar{B} C D$

Chapter 10

1.

		0	1
00	1	1	2
001	2	—	3
0011(11)	③	4	3
0011(11)00	④	4	—

2.

		0	1
00	1	1	2
001	②	4	3
0011	3	—	4
00100 } 00111(00) }	④	4	—

4.

		0	1
——00	①	1	2
——001	②	1	3
——0011	3	—	4
——00111	④	1	—

6.

		0	1
00	1	1	2
001—odd—1	②	4	3
001—even—1	3	—	2
001—odd—100	④	4	—

7.

		0	1
11	1	2	1
110	②	3	3
11011 } 1100(11) }	③	—	3

8.

		0	1
00	1	1	2
001	2	—	3
0011	3	4	—
001100	4	4	5
0011001	5	—	6
00110011(00)	⑥	6	—

10.

		0	1
—00	1	1	2
—001	2	1	3
—0011	3	1	4
—00111	④	6	5
—001111(11)	5	1	5
—0011100—00	⑥	6	7
—0011100—001	⑦	6	8
—0011100—0011	⑧	6	9
—0011100—00111	9	—	10
—0011100—001111(11)	⑩	6	10

11.

		0	1
00	1	1	2
001	2	3	4
00100	3	3	5
0011 } 0010011 }	4	—	6
001001	5	—	4
00111(00) } 00100111(00) }	⑥	6	—

12.

		0	1
00	1	1	2
001	2	—	3
0011	3	4	7
001100	4	4	5
0011001	5	—	6
00110011	6	—	9
00111	7	8	—
0011100	8	8	9
001100111(00) } 00111001(00) }	⑧	9	—

Chapter 11

1. $1 \times 1 = 1$

$1 \times 2 = 2$

$0 \times 4 = 0$

$1 \times 8 = 8$

$1 \times 16 = 16$

$1 \times 32 = 32$

59 (base 10)

$$\begin{array}{r}
 3 \overline{)59} \quad 2 \uparrow \\
 3 \overline{)19} \quad 1 \\
 3 \overline{)6} \quad 0 \\
 3 \overline{)2} \quad 2 \\
 \quad 0
 \end{array}$$

2012 (base 3)

$$\begin{aligned}
 2. \quad & 1 \times 1 = 1 \\
 & 0 \times 7 = 0 \\
 & 6 \times 49 = 294 \\
 & 2 \times 343 = 686
 \end{aligned}$$

981 (base 10)

$$\begin{array}{r|l}
 6 & \overline{981} & 3 \uparrow \\
 6 & \overline{163} & 1 \\
 6 & \overline{27} & 3 \\
 6 & \overline{4} & 4 \\
 & 0 &
 \end{array}$$

4313 (base 6)

$$\begin{array}{r|l}
 4. \quad 2 & \overline{13} & 1 \uparrow \\
 2 & \overline{6} & 0 \\
 2 & \overline{3} & 1 \\
 2 & \overline{1} & 1 \\
 & 0 &
 \end{array}$$

$$\begin{array}{r|l}
 & .8125 \\
 & \times 2 \\
 1 & .6250 \\
 & \times 2 \\
 1 & .2500 \\
 & \times 2 \\
 0 & .5000 \\
 & \times 2 \\
 1 & .0000
 \end{array}$$

1101.1101 (base 2)

Chapter 12

1.

	1	2	3	4	5	6	7	8	
	C_1	C_2	8	C_4	4	2	1	P	
	0	1	0	1	0	1	0	1	→ 2
	1	0	0	0	0	1	1	1	→ 3
	1	0	0	1	1	0	0	1	→ 4
	0	1	0	0	1	1	1	0	→ ? (Double error, no correction made)*

*Note that this digit could be a 5, 6, or 7:

$$\begin{array}{rcl}
 0 & 1 & 0 & 0 & 1 & \mathbf{0} & 1 & \mathbf{1} & \rightarrow 5 \\
 \mathbf{1} & 1 & 0 & 0 & 1 & 1 & \mathbf{0} & 0 & \rightarrow 6 \\
 0 & \mathbf{0} & 0 & \mathbf{1} & 1 & 1 & 1 & 0 & \rightarrow 7
 \end{array}$$

Chapter 14

1.

$x_1 \ x_2$						Z
00	01	11	10			
①	4	5	7	0		0
②	4	6	8	1		1
2	③	5	8	0		0
1	④	6	8	1		1
1	3	⑤	8	0		0
2	4	⑥	7	1		1
1	4	6	⑦	0		0
2	3	6	⑧	1		1

Figure 14-1A

2.

$x_1 x_2$		00	01	11	10	Z
①	3	4	6	0		0
1	②	4	6	0		0
1	③	4	6	1		1
-	2	④	5	1		1
1	2	4	⑤	0		0
1	2	4	⑥	1		1

Figure 14-2A

4.

$$1 \equiv 3$$

$$4 \equiv 6$$

$$8 \equiv 9$$

$$10 \equiv 11$$

$x_1 x_2$		00	01	11	10	Z
①	4	8	10	0		0
②	4	7	10	0		0
1	④	8	10	1		1
1	⑤	8	12	1		1
1	5	⑦	10	0		0
1	4	⑧	10	0		0
1	4	8	⑩	1		1
2	4	8	⑫	0		0

Figure 14-4A

6.

$$5 \equiv 7$$

$$6 \equiv 7$$

$$1 \equiv 2$$

$$3 \equiv 4$$

$x_1 x_2$		00	01	11	10	$Z_1 Z_2$
①	-	3	5	00		00
1	-	③	6	11		11
1	8	3	⑤	10		10
1	9	3	⑥	10		10
1	⑧	3	-	01		01
1	⑨	3	-	10		10

Figure 14-6A

Chapter 15

1.

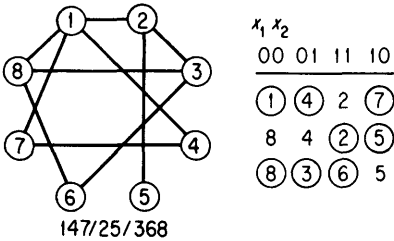


Figure 15-1A

Chapter 16

1.

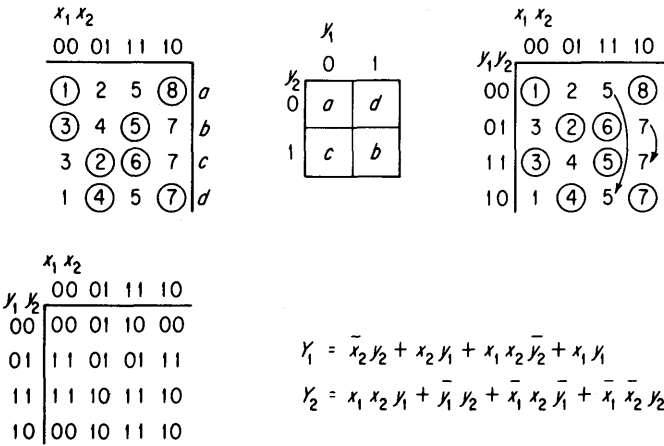


Figure 16-1A

3.

			$x_1 x_2$			
			00	01	11	10
y_1	y_2	y_3				
0	0	0	000	010	001	100
0	0	1	001	000	101	000
0	1	1	---	---	---	010
0	1	0	110	010	010	010
1	1	0	110	111	010	110
1	1	1	101	111	111	011
1	0	1	001	101	101	111
1	0	0	---	---	---	110

Figure 16-3A

5. $Y_1 = x_1 x_2 \bar{y}_2 + \bar{x}_1 \bar{x}_2 y_2 + y_1 y_2$ or $(x_1 + y_2)(\bar{x}_1 + x_2 + y_1)(\bar{x}_2 + y_1 + \bar{y}_2)$
 $Y_2 = x_1 \bar{x}_2 + \bar{y}_1 y_2 + x_1 y_1 + \bar{x}_2 y_1$ or $x_1 \bar{x}_2 + \bar{y}_1 y_2 + x_1 y_1 + \bar{x}_2 y_2$
 or $(\bar{x}_2 + y_1 + y_2)(x_1 + \bar{x}_2 + \bar{y}_1)(x_1 + y_2)$

Note that the product of sums solutions, with $(x_1 + y_2)$ common to both Y_1 and Y_2 , are optimum.

Chapter 17

1.

		$x_1 x_2$				
		00	01	11	10	
y_1	y_2					
0	0	10	11	01	01	$Z_1 = \bar{x}_1 \bar{y}_1 + x_1 \bar{y}_2$
0	1	1-	11	0-	01	$Z_2 = x_1 \bar{y}_1 + x_2 \bar{y}_2 + \bar{y}_1 \bar{y}_2$
1	1	00	--	00	-0	or $x_1 \bar{y}_1 + x_2 \bar{y}_2 + \bar{x}_1 x_2$
1	0	-0	11	-1	10	or $x_1 \bar{y}_1 + x_2 \bar{y}_2 + x_2 \bar{y}_1$

Z-map

Figure 17-1A

3.

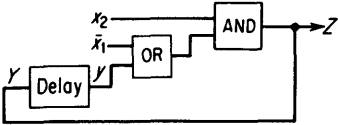
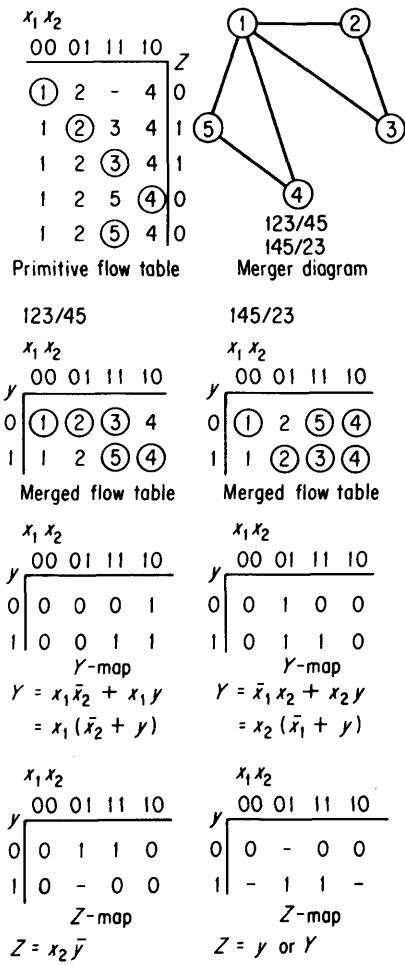


Figure 17-3A

Chapter 18

1.

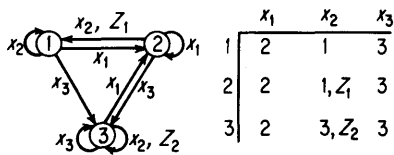


Figure 18-1A

2.

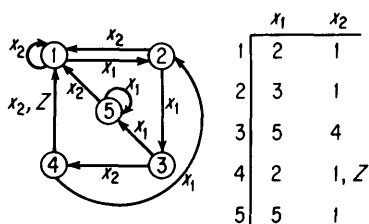


Figure 18-2A

3.

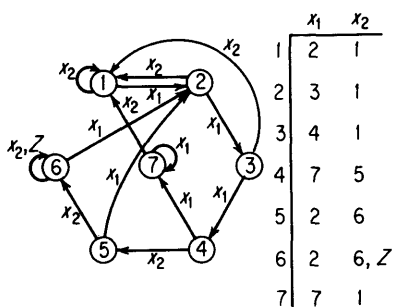


Figure 18-3A

Chapter 19

1.

$y_1 y_2$	x_1	x_2	Z
00	10	00	0
10	01	11	1
01	01	00	0
11	10	11	1

y_2	y_1	x_1	x_2
0	0	1	0
1	0	1	1

y_2	y_1	x_1	x_2
0	0	0	1
1	0	0	1

y_2	y_1	x_1	x_2
0	0	0	1
1	0	0	1

y_2	y_1	x_1	x_2
0	0	0	1
1	0	0	1

y_2	y_1	x_1	x_2
0	0	0	1
1	0	0	1

Figure 19-1A

3.

\overline{FF}	Inputs	
$S-R$	$S_1 = x_1 y_2 + x_2 \bar{y}_1 y_2$ $R_1 = x_1 \bar{y}_2 + x_2 y_1 y_2$	$S_2 = x_2 \bar{y}_1 \bar{y}_2$ $R_2 = x_2 \bar{y}_1 y_2$
$S-R-SR$	$S_1 = x_1 y_2 + x_2 y_2$ $R_1 = x_1 \bar{y}_2 + x_2 y_2$	$S_2 = x_2 \bar{y}_1$ $R_2 = x_2 \bar{y}_1$
T	$T_1 = x_1 \bar{y}_1 y_2 + x_1 y_1 \bar{y}_2 + x_2 y_2$	$T_2 = x_2 \bar{y}_1$
$S-R-T$	$S_1 = x_1 y_2$ $R_1 = x_1 \bar{y}_2$ $T_1 = x_2 y_2$	$S_2 = \text{---}$ $R_2 = \text{---}$ $T_2 = x_2 \bar{y}_1$
$S-R-SR-T$	$S_1 = x_1 y_2$ $R_1 = x_1 \bar{y}_2$ $T_1 = x_2 y_2$	$S_2 = \text{---}$ $R_2 = \text{---}$ or $R_2 = x_2 \bar{y}_1$ $T_2 = x_2 \bar{y}_1$ $T_2 = \text{---}$

$$Z = y_2$$

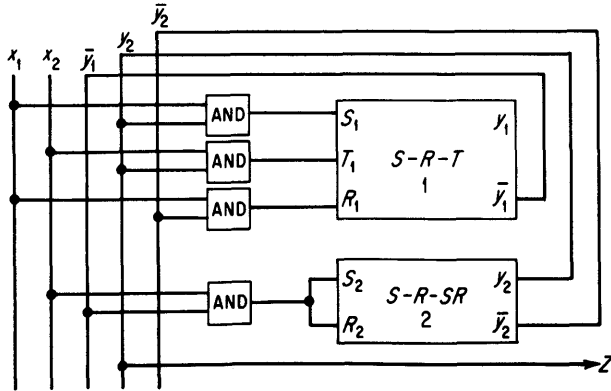


Figure 19-3A

4.(a)

	x_1	x_2		$y_1 y_2$	x_1	x_2
1	2	1	00	01	00	
2	2	3	01	01	11	
3	2	1, Z	11	01	00, Z	
			10	--	--	

y_2	y_1	x_1
0	0	1
0	0	-
1	0	0

y_2	y_1	x_2
0	0	1
0	0	-
1	1	0

FF_1

y_2	y_1	x_1
0	0	1
0	1	-
1	1	1

y_2	y_1	x_2
0	0	1
0	0	-
1	1	0

FF_2

$S-R-T$ $S_1 = \text{---}$
 $R_1 = x_1$
 $T_1 = x_2 y_2$

$S-R$ $S_2 = x_1$
 $R_2 = x_2 y_1$

$Z = x_2 y_1$

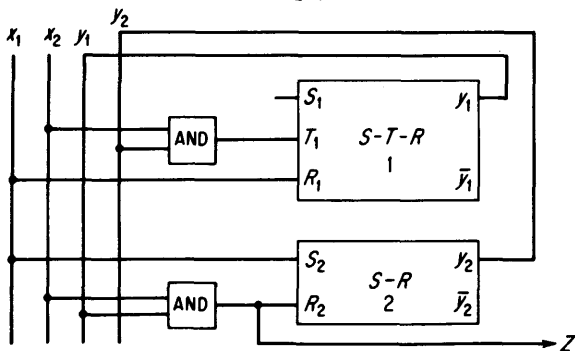


Figure 19-4A

(b)

	x_1	x_2		$y_1 y_2$	x_1	x_2
1	2	1	00	01	00	
2	2	3	01	01	10	
3	2	1, Z	10	01	00, Z	
			11	--	--	

	y_1	x_1
y_2	0	1
0	0	0
1	0	-

	y_1	x_2
y_2	0	1
0	0	0
1	1	-

FF_1

	y_1	x_1
y_2	0	1
0	1	1
1	1	-

	y_1	x_2
y_2	0	1
0	0	0
1	0	-

FF_2

$$\begin{aligned}
 S-R-T \quad S_1 &= x_2 y_2 \\
 R_1 &= x_1 \\
 T_1 &= x_2 y_1
 \end{aligned}$$

$$\begin{aligned}
 S-R \quad S_2 &= x_1 \\
 R_2 &= x_2
 \end{aligned}$$

$$Z = x_2 y_1$$

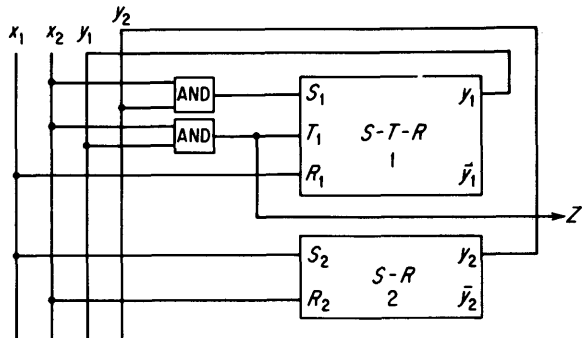


Figure 19-4A

(c)

	x_1	x_2		$y_1 y_2$	x_1	x_2
1	2	1	00	11	00	
2	2	3	11	11	01	
3	2	1, Z	01	11	00, Z	
			10	--	--	

	y_1	x_1
y_2	0	1
0	1	-
1	1	1

	y_1	x_2
y_2	0	1
0	0	-
1	0	0

FF_1

	y_1	x_1
y_2	0	1
0	1	-
1	1	1

	y_1	x_1
y_2	0	1
0	0	-
1	0	1

FF_2

$S-R$ $S_1 = x_1$
 $R_1 = x_2$

$S-R$ $S_2 = x_1$
 $R_2 = x_2 \bar{y}_1$

$Z = x_2 \bar{y}_1 y_2$

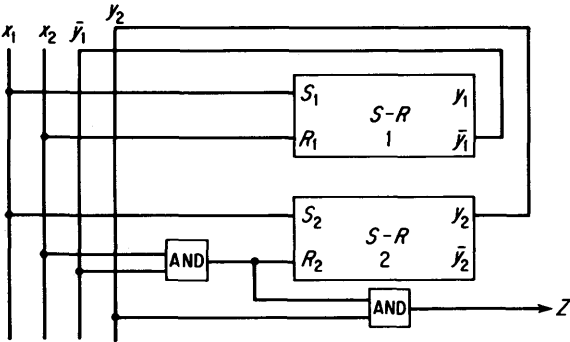


Figure 19-4A

5.

	x_1	x_2		$y_1 y_2$	x_1	x_2
1	2	1	00	01	00	
2	3	1	01	11	00	
3	3	1, Z	11	11	00, Z	
			10	--	--	

	y_1	x_1
y_2	0	1
0	0	-
1	1	1

	y_1	x_2
y_2	0	1
0	0	-
1	0	0

FF_1

	y_1	1
y_2	0	1
0	1	-
1	1	1

	y_1	1
y_2	0	1
0	0	-
1	0	0

FF_2

$$S-R \quad S_1 = x_1 y_2$$

$$R_1 = x_2$$

$$S-R \quad S_2 = x_1$$

$$R_2 = x_2$$

$$Z = x_2 y_1$$

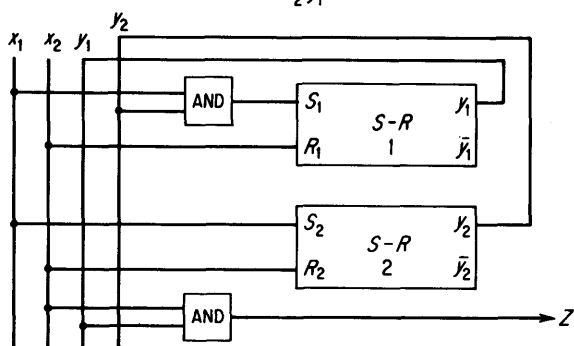


Figure 19-5A

INDEX

A

- Adders, 161
 - BCD, 165
 - full-, 163
 - half-, 161
- Addition, 9 (*see also* OR)
- Algebra, Boolean (*see* Boolean algebra)
- Alphanumeric codes, 179
- Alternative sequences, 189
- AND, 2
 - circuit, 39, 42, 45
 - diode, 48
 - transistor, 53
 - function, 37, 45
- Assignment, secondary state (*see* Secondary state assignment) (*see also* Spare secondary states)

B

- BCD adder, 165
- BCD code, 165, 168
- Binary adders (*see* Adders)
- Binary-coded-decimal adder, 165
- Binary-coded-decimal code, 165, 168
- Binary number system, 160
- Binary ordering, 99, 105
 - reflected, 98, 105
- Biquinary code, 176
- Boolean algebra, 1

Boolean algebra (*cont.*):

- method of attack, 18
- postulates, 5
 - summary, 23
- theorems, 7
 - summary, 23
- Boolean expressions, special forms (*see* Special forms of Boolean expressions)
- Boolean operations with symmetric notations, 128
- Break-before-make contacts, 61
- Bridge circuits, 63

C

- Canonical form, 30
- Chart, timing, 186
- Chart, Veitch, 105
- Codes:
 - alphanumeric, 179
 - BCD, 165, 168
 - biquinary, 176
 - cross-parity, 180
 - cyclic, 169
 - excess-3, 168
 - fixed bit, 175, 180
 - Gray, 169
 - Hamming, 176, 179
 - m*-out-of-*n*, 175, 180
 - numeric, nonchecking, 167

Codes (*cont.*):

- parity, 174
 - cross-, 180
- quibinary, 176
- reflected BCD, 170
- reflected binary, 169
- reflected excess-3, 170
- single-error correction, 176
 - with double-error detection, 179
- single-error detection, 174
- two-out-of-five, 175

Combinational circuit, 36

Complement, 6

Complementary approach:

- map method, 102
- tabular method, 85

Complementation, contact network, 64

Conjunctive normal form, 30

Contact networks, 57

- bridge circuits, 63
- and complement, 65
- complementation, 64
- implementation of AND, OR, and NOT, 59

- multi-output, 65

- nonplanar networks, 63

- series-parallel, 60

- symmetric, 129

- transfer contacts, 61

Contacts, 58

- parallel path, 60

- series path, 59

- transfer, 61

Continuity-transfer contact, 61

Correction, error, 171 (*see also* Codes)

Critical race, 211

Cross-parity, 180

Cycles, 209

Cyclic codes, 169

Cyclic specifications, 241

D

Delay, 182

DeMorgan's theorem, 10

Detection error, 171 (*see also* Codes)

Detection and identification of symmetric functions, 137

- outline, 138

Diagram, flow, 259, 261

Diagram, merger, 206

Diode:

- AND circuit, 48
- logic blocks, 48
- OR circuit, 49

Disjunctive normal form, 30

Distance, 171

- minimum, 171

Don't care combination, 72

Dotting, 53

Dual, 7

E

Electronic logic blocks (*see* Logic blocks)

Electronic trees, 119

- most economical, 119

Elimination of redundant input lines, re-iterative networks, 151

Elimination of redundant states, 197, 266

Emitter follower, 51

Equivalence, 197, 266

- pseudo-, 200, 266

Equivalent expressions, 6

Equivalent states (*see* Equivalence)Error detection and correction, 171 (*see also* Codes)

Essential prime implicant, 78

Even parity code, 174

Excess-3 code, 168

Excitation, secondary, 182

Excitation map:

- flip flop (*see* Flip flop excitation maps)

- secondary, 190

Exclusive OR, 4, 21, 46

Expanded product of sums, 27, 28

Expanded sum of products, 27, 28

F

Factoring on map, 109

Feedback path, 183

Fixed bit codes (*see m-out-of-n codes*)

Flip flops, 253

- excitation maps, 273

- entries, 274

- reading, 278

- summary, 294

S-R, 253, 278

S-R-SR, 281

Flip flops (*cont.*):*S-R-SR-T*, 291*S-R-T*, 287*T*, 255, 284

Flow diagram, 259, 261

Flow table, 190, 260, 261

incompletely specified, 203

merged, 206

primitive, 194

Follower, emitter, 51

Full-adder, 163

Functions of n variables, 33

G

Gain, 183

Graphical complementation, 64

Gray code, 169

H

Half-adder, 161

Hamming code, 176, 179

Hazards, 243

identification in map, 245

I

Implicants, prime (*see* Prime implicants)

Included factor theorem, 15

Included term theorem, 15

Incompletely specified flow table, 203

Induction, perfect, 8, 14

Intuitive approach, sequential circuit synthesis, 185

Invalid combination, 72

Inverter (*see* NOT circuit)

Iterative method for obtaining prime implicants, 87

Iterative network (*see* Reiterative network)

K

Karnaugh map, 105 (*see also* Map method of simplification)

L

Literal, 6

Logic (*see* Mixed logic, Negative logic, Positive logic)

Logic blocks, 37, 38

electronic, 48

diode, 48

transistor, 51

vacuum tube, 49

Logical circuits, 36 (*see also* Logic, Logic blocks)

M

Make-before-break contact, 61

Map:

flip flop excitation (*see* Flip flop excitation maps)

method of simplification, 97

complementary approach, 102

factoring on map, 109

maps of more than four variables, 106

method of attack, 103

with optional combinations, 106

summary, 109

transition, 212

Y-, 190, 212, 214*Z*-, 190, 236

Minterm, 30

Memory, 182

Merged flow table, 206

Merger diagram, 206

Mesh, 64

Minimization of partial trees, 114

Minimum distance, 171

Minimum factored form, 33

Minimum product of sums, 27, 30

Minimum sum of products, 27, 30

Minterm, 30

Mixed logic, 45

 m -out-of- n codes, 175, 180 m -out-of- n functions, 127

Multi-output networks, 88

contact, 65

like contacts, parallel paths, 67

like contacts, series paths, 66

network and complement, 65

Multiple-output prime implicants, 88

Multiple secondary states to a row, 230

patterns, 232

Multiplication, 9 (*see also* AND)

Multivibrator, 255

single-shot, 255



N**NAND:**

- circuit, 40, 43, 45
- transistor, 53
- vacuum tube, 50
- function, 37, 38, 45-47

Negative logic, 41

- application, 44

Node, 64**Noncritical race, 210****Nonequivalence, 199**

- tabular approach, 199

Nonplanar networks, 63**NOR:**

- circuit, 40, 43, 45
- transistor, 53
- vacuum tube, 50
- function, 37, 38, 45-47

Normal form, 30**NOT, 4**

- circuit, 41, 44
- transistor, 51
- vacuum tube, 49
- function, 37

Number systems, 156

- binary, 160

Numeric codes, nonchecking, 167**O****Odd parity code, 174****One-shot multivibrator, 255****Optional combinations, 72**

- with map method, 106
- with tabular method, 79

OR, 3

- circuit, 40, 43, 45
- diode, 49
- transistor, 53
- vacuum tube, 50
- exclusive, 4, 21, 46
- function, 37, 45

Ordering (see Binary ordering)**Outputs, transient, 241****P****Parallel paths:**

- contacts in, 60
- like contacts in, 67

Parity, cross-, 180**Parity code, 174****Partial trees, minimization, 114****Patterns, spare secondary states (see Spare secondary states, patterns)****Perfect induction, 8, 14****Pierce Arrow function, 38****Position, relay, 61****Positional circuit, 145****Positive logic, 39**

- application, 44

Postulates, Boolean algebra, 5

- summary, 23

Power-on output state, 196**Primary, 182****Prime implicants, 74**

- essential, 78
- iterative method, 87
- multiple-output, 88
- weighting, 82

Primitive flow table, 194**Product, 9 (see also AND)**

- canonical, 30
- standard, 30

Product of sums, 9

- expanded, 27, 28
- minimum, 27, 30

Prototype relay, 146**Pseudo-equivalence, 200, 266****Pulse, 252****Pulse-input sequential circuits, 252, 256**

- elimination of redundant states, 266
- flip flops (see Flip flops)
- flow diagram, 259, 261
- flow table, 260, 261
- most-economical circuit, 298
- output, 295
- secondary assignment, 269
- synthesis, 258

Q**Quibinary code, 176****R****Races, 210**

- critical, 211
- noncritical, 210

Radical point, 156**Redundant input lines, reiterative networks, 151**

- Redundant states, 197, 266
- Reflected BCD code, 170
- Reflected binary code, 169
- Reflected binary ordering, 98, 105
- Reflected excess-3 code, 170
- Reiterative networks, 145
 - redundant input lines, 151
 - sequence representation, 150
- Relay contact networks (*see* Contact networks)
- Relay contacts (*see* Contacts)
- Relay trees, 112
 - minimization of partial, 114

- S**
- Secondary, 182
 - excitation, 182
 - excitation map, 191
 - state assignment, 212, 269 (*see also* Spare secondary states)
- Sequential circuits, 182 (*see also* Pulse-input sequential circuits)
 - basic operation, 185
 - cycles 209
 - cyclic specifications, 241
 - elimination of redundant states, 197
 - flow table (*see* Flow table)
 - hazards, 243
 - merger diagram, 206
 - most-economical circuit, 247
 - power-on output state, 196
 - pseudo-equivalence, 200
 - races, 210
 - secondary state assignment, 212
 - spare secondary states (*see* Spare secondary states)
 - stability, 184
 - synthesis, 193
 - intuitive approach, 185
 - transient outputs, 241
 - transition map, 212
 - Y-map, 190, 212, 214
 - Z-map 190, 236
- Series-parallel contact network, 60
- Series paths:
 - contacts in, 59
 - like contacts in, 66
- Set-reset flip flop, 253, 278
- Sheffer Stroke function, 38
- Shift down, 133
 - with three or more subscripts, 136
- Simplification:
 - map method, 97
 - tabular method, 71, 73
 - theorems, 18
- Single-error correction codes, 176
 - with double-error detection, 179
- Single-error detection codes, 174
- Single-shot multivibrator, 255
- Spare secondary states, 217
 - assignments, 222
 - multiple secondary states to a row, 230
 - patterns, 220, 232
 - summary, 233
- Special forms of Boolean expressions, 27
 - expanded product of sums, 28
 - expanded sum of products, 28
 - minimum expressions, 30
 - minimum factored form, 33
- Spring, relay, 61
- S-R flip flop, 253, 278
- \bar{S} -R-SR flip flop, 281
- S-R-SR-T flip flop, 291
- S-R-T-flip flop, 287
- Stability, 184
- Stable secondary, 184
- Standard sum and product, 30
- State, secondary, 182
- State assignment (*see* Secondary state assignment) (*see also* Spare secondary states)
- States, equivalent, 197, 266
- States, redundant, 197, 266
- Subsume, 13
- Sum, 9 (*see also* OR)
 - canonical, 30
 - standard, 30
- Sum of products, 9
 - expanded, 27, 28
 - minimum, 27, 30
- Symmetric functions, 126
 - Boolean operations, 128
 - contact networks, 129 (*see also* Symmetric relay contact networks)
 - in design of electronic switching circuits, 161
 - detection and identification, 137
 - outline, 138
 - m -out-of- n functions, 127
 - variables of symmetry, 126

- Symmetric relay contact networks, 129
 - identification of transfer contacts, 131
 - symmetric circuits with multiple m 's, 131
 - complementation, 135
 - complemented variables of symmetry, 136
 - elimination of redundant transfer contacts, 132
 - equivalent points, 134
 - shift down, 133
 - symmetric tree, 130
- Symmetry, variables of (*see* Variables of symmetry)
- T**
 - T flip flop, 255, 284
 - Table, flow (*see* Flow table)
 - Table, truth, 14
 - Tabular method of simplification, 71, 73
 - algebraic solution of final table, 80
 - complementary approach, 85
 - with optional combinations, 79
 - weighting prime implicants, 82
 - Theorems, Boolean algebra, 7
 - summary, 23
 - Timing chart, 186
 - Transfer contacts, 61
 - Transfer trees (*see* Relay trees)
 - Transient outputs, 241
 - Transistor, 51
 - AND circuit, 53
 - emitter-follower, 51
 - inverter, 51
 - logic blocks, 51
 - NAND circuit, 53
 - NOR circuit, 53
 - NOT circuit, 51
 - OR circuit, 53
 - Transition map, 212
 - Transposition theorems, 20
 - Trees:
 - electronic, 119
 - most-economical, 119
 - relay, 112
 - minimization of partial, 114
 - symmetric, 130
 - Trigger, 255, 284
 - Truth table, 14
 - Two-out-of-five code, 175
- U**
 - Unstable secondary, 184
- V**
 - Vacuum tube:
 - inverter, 49
 - logic blocks, 49
 - NAND circuit, 50
 - NOR circuit, 50
 - NOT circuit, 49
 - OR circuit, 50
 - Variables, 6
 - Variables of symmetry, 126
 - complemented, 129, 136
 - Veitch chart, 105
- W**
 - Weighting prime implicants, 82
- Y**
 - Y -map, 190, 212, 214
- Z**
 - Z -map, 190, 236

(Continued from front flap)

vantage of this method is that excitation expressions for any type of flip-flop can be read from a single map set.

- There is a strong emphasis on sequential circuits—almost 50 per cent of the book.
-

MITCHELL P. MARCUS is a Senior Engineer with the IBM Corporation. In addition to his work in switching circuit theory and its application to the logical design of IBM products, he has been teaching courses in switching circuits at IBM since 1954. He has had many publications in professional journals, has been granted numerous patents, and has received an IBM Outstanding Invention Award.

PRENTICE-HALL, Inc.
Englewood Cliffs, New Jersey

Other recommended books of interest

DIGITAL COMPUTER ENGINEERING

by HARRY J. GRAY, *University of Pennsylvania*

This book is directed at the analytical and practical problems which must be systematically solved in designing a high-speed digital-computer system. It treats digital circuit theory, signal transmission and noise, statistical design, and the integration of these into the digital computer design practice.

It includes cross-talk prediction in a computing system, circuit analysis techniques that are of practical application, circuit synthesis techniques that have been found to be of value, statistical design, reliability consideration peculiar to digital computers, logical requirements for digital computer circuits, statement of the problem of synthesis for digital systems, general characteristics of synchronous and asynchronous systems, and areas where design automation has been of value.

Published 1963

381 pages

ALGEBRAIC STRUCTURE THEORY OF SEQUENTIAL MACHINES

by J. HARTMANIS, *Cornell University* and

R. E. STEARNS, *General Electric Research and Development Center*

The authors present the first thorough treatment of the structure theory of sequential machines and its applications to machine synthesis and machine decomposition into smaller component machines. The unified mathematical approach developed by the authors produces results that are easily understandable and directly applicable to the design of sequential machines.

All the structure and decomposition results, including those derived from semi-group analysis, are obtained through the application of partition algebra and its generalizations. The mathematical formalization expresses algebraically the intuitive concept of information, and makes possible the solution of problems related to the flow of this information in machines.

Published 1966

211 pages

COMPUTATION: FINITE AND INFINITE MACHINES

by MARVIN MINSKY, *Massachusetts Institute of Technology*

Provides an introduction to the theories of finite-state machines, programmed computers, Turing machines and formal languages (in the form of Post Systems). Some of the outstanding features include: • topics covered range from basic principles to current research problems • extensive discussion of the meaning and motivation of the theory, its practical value and limitations. Brings together three different approaches: Neural Nets (of interest to Life Scientists), Turing Machines and abstract languages, which are usually treated as different, disconnected topics.

Published 1967

320 pages

PRENTICE-HALL, Inc.
Englewood Cliffs, New Jersey



SWITCHING CIRCUITS
FOR ENGINEERS
SECOND EDITION

Marcus

621.316
.5.049
M

PRENTICE
HALL

87986